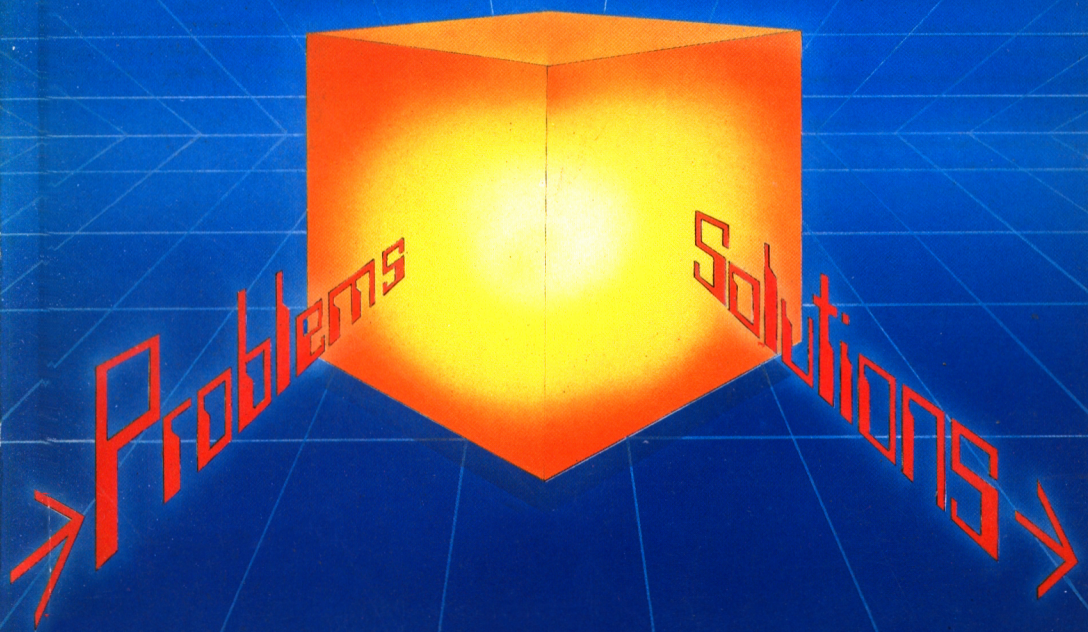
POWERFUL PROGRAMMING FOR Amstrads

SUPERCHARGE YOUR 464/664/6128



W. Johnson

SIGMA PRESS

Powerful Programming for Amstrads

– supercharge your 464/664/6128

W. Johnson

Σ
SIGMA
PRESS

ISBN 1 85058 046 4

Published by:

Sigma Press
98a Water Lane
Wilmslow
Cheshire
U.K.

Distributed by:

U.K., Europe, Africa:

John Wiley & Sons Ltd
Baffins Lane
Chichester
West Sussex
U.K.

Australia:

John Wiley & Sons Inc
GPO Box 859,
Brisbane
Queensland 4001
Australia

Printed by:

Interprint Ltd, Malta

PREFACE

This collection of subroutines has been compiled for publication from a set of programs written in BASIC to solve problems in engineering, image manipulation and, data handling and storage. It should be of interest to all computer owners as it contains useful reference routines as well as routines which are hard to find or tedious to write.

The subroutines are written in Amstrad BASIC but can easily be translated into other dialects of BASIC and many of the programs are worth studying in detail for the techniques used. There is great satisfaction to be had from producing an elegant solution to a problem and the underlying theme is that a program should be a body of instructions with efficient subroutines. The efficiency should be in terms of the time of execution, the memory requirements, freedom from crashing and the accuracy of the result, though presentation of the results is also important.

A good programmer is one who is not easily satisfied with a program and will always seek ways to improve it.

Part of the pleasure of computing, after the novelty of playing games has worn off, is to develop your own programs and games and make the computer do what you want with the minimum number of instructions and memory requirements. To accomplish this you need techniques for generating and handling data efficiently as well as being able to store and retrieve them quickly. Good visual presentation of results considerably enhances the program and the listings should be well documented and easy to follow.

This is the aim of the book, which contains over a hundred subroutines to help you program better. Every effort has been made to ensure that the subroutines work over the ranges specified and in an efficient way but there are no prizes for finding cases where they do not work.

At the end of the book, a few complete programs are given to illustrate the use of some of the subroutines. They include drawing crystal shapes, solving an anagram, studying the stability of an automatic control system and imaging a triangle of any shape, as well as a very efficient storage and retrieval program.

CONTENTS

The Subroutines

The majority of the subroutines are set out in three sections. The middle section is the subroutine itself which begins at line 1000 to assist you in entering it as a subroutine. The first section is a simple input routine to enable the subroutine to be checked after typing it in. The final section always begins at 2000 and is a simple output routine to be used in the same way as the input one. All three can be used together if you wish to use the program as a stand–alone routine.

Where a DIM statement occurs in the subroutine itself, it is normally ERASEd at the end of the output section to enable the subroutine to be used again. Where a DIM statement occurs in the input section, it is assumed that it is part of the main program and that it will carry the information back with the RETURN.

For good readability, the following conventions have been adopted. Reserved words and colons are followed by a space. Mathematics are set out without spaces and, if arguments have to be enclosed in brackets, these take the place of the space after the reserved word. Integers are usually defined but the symbols % and ! have not been used in the program variable names except in a few cases. Strings are almost entirely defined in this way rather than using DEFSTR at the beginning. This is to make the program more understandable to readers unfamiliar with this facility. SW or SW$ is used for the dummy in swapping operations. Square brackets have been used in array names.

Care should be taken to ensure that the variables used do not conflict with those in the main program either in type or precision level. A,B,C,H,K,L, M,N,P,Q,R,S,T,X,Y,Z are the most common ones.

A brief explanation of how each subroutine works is given and usually, an example is shown of a typical output.

After entering, the subroutine can be checked and then placed where required with RENUM. Alternatively, the subroutines can be recorded and brought in as necessary with MERGE, provided any lines 1000 onwards have been shifted to safety with RENUM.

1. ADJECTIVAL NUMBER ENDINGS

This subroutine enables "st", "nd", "rd" or "th" to be printed after a number as appropriate. The Defined Function either prints A$ (i.e. "st" etc.) if the conditions are true or a null string if the conditions are false. The variable BOOLE can only take the values of −1 (true) or 0 (false).

```
10 REM Adjectival Number Endings
20 DEF FN P$(A$,BOOLE)=MID$(A$,1,-LEN(A$)*BOOLE)
30 N=0: WHILE N<=0 OR N>999999999: INPUT N: WEND
40 IF N<>INT(N) THEN 30
50 N$=MID$(STR$(N),2)
60 GOSUB 1000: END
1000 IF LEN(N$)=1 THEN PRINT N$;FN P$("st",N=1)+FN P$("nd",N
=2)+FN P$("rd",N=3)+FN P$("th",N<>1 AND N<>2 AND N<>3): RETU
RN
1010 L$=RIGHT$(N$,1): NL$=RIGHT$(N$,2): PRINT N$;FN P$("st",
L$="1" AND NL$<>"11")+FN P$("nd",L$="2" AND NL$<>"12")+FN P$
("rd",L$="3" AND NL$<>"13")+FN P$("th",(L$<>"1" AND L$<>"2"
AND L$<>"3" OR (NL$="11" OR NL$="12" OR NL$="13"))): RETURN
```

2. ANNUITIES CERTAIN

These subroutines work out the annuity certain tables £$A_{\overline{n}|}$ for different annual interest rates and for annual and monthly payments.

Financial transactions based on compound interest have the geometric series underlying them. If i is the interest rate, then £1 will become £$(1+i)^n$ in n years time so that, turning it over £$1/(1+i)^n$ will become £1 in n years time. The annuity certain is the sum of the present values $1/(1+i)^n$ for each of the years to come.

Hence, $A_n = v+v^2+v^3+v^4 \ldots\ldots v^n$ where $v=1/(1+i)$

This series equals $(1-v^n)/i$ which has been used to calculate the tables shown.

```
10 REM Annual Payments
20 N=0: WHILE N<=0: INPUT "Number of years";N: WEND
30 DIM A[N]
1000 INPUT "Annual Interest Rate as %";I
1010 GOSUB 1020: GOSUB 1040: END
1020 FOR P=1 TO N: A[P]=(1-(100/(100+I))^P)/I*100: NEXT
1030 RETURN
1040 FOR P=5 TO N: PRINT USING "###.#####";A[P]: NEXT
1050 RETURN
```

```
10 REM Monthly Payments
20 N=0: WHILE N<=0: INPUT "Number of years";N: WEND
30 DIM A[N]: GOSUB 1000: GOSUB 2000: END
1000 INPUT "Annual Interest Rate as %";I: I=I/12
1010 FOR P=1 TO N: A[P]=(1-(100/(100+I))^(P*12))/I*100: NEXT
1020 RETURN
2000 FOR P=10 TO N: PRINT P, USING "###.#####";A[P]: NEXT
2010 RETURN
```

The annuity certain is the initial sum which at x% per annum yields £1 per annum over n years. If you have for example £25,000 to invest at 8% pa and want to draw 15 equal annual instalments, then the annuity certain for 15 years at 8% is 8.5595 and £25,000/8.5595=£2,920.74 is the annual instalment. After 15 years the money is all used up, as you have been drawing capital and interest.

The reverse example is paying off loans, e.g. a mortgage, so that the monthly

payments are constant, i.e. initially you pay mainly interest charges, but gradually pay back more capital.

MONTHLY REPAYMENTS AT 9%

Years	*Annuity Certain*
10	78.94169
11	83.60642
12	87.87109
13	91.77002
14	95.33457
15	98.59341
16	101.57277
17	104.29661
18	106.78686
19	109.06353
20	111.14495

ANNUAL INSTALMENTS AT 8%

Years	*Annuity Certain*
5	3.99271
6	4.62288
7	5.20637
8	5.74664
9	6.24689
10	6.71008
11	7.13896
12	7.53608
13	7.90378
14	8.24424
15	8.55948
16	8.85137
17	9.12164
18	9.37189
19	9.60360
20	9.81815

3. BEST FIT LINE

This subroutine calculates the best fit straight line for a set of points and, plots the points and the line with appropriately scaled and labelled axes. It prints out the slope and intercept of the line and the correlation coefficient for the data. Dotted lines are drawn to represent the 95% confidence limits which are also shown numerically.

The first 15 lines work out the mathematical quantities needed from the INPUT data. These are the maximum values of x and y (MX,MY) to scale the graph properly and the mean values (XM,YM) from which the slope and intercept of the best fit line and, the variance can be calculated.

The best fit line is given by

$$y = \frac{\Sigma (x_i-\bar{x})(y_i-\bar{y})}{\Sigma (x_i-\bar{x})^2} .x + \frac{\Sigma y_i}{n} - \frac{\Sigma x_i . \Sigma (x_i-\bar{x})(y_i-\bar{y})}{n. \Sigma (x_i-\bar{x})^2}$$

where the sum is over values of i from 1 to N.

The regression coefficient is given by

$$r = \frac{\Sigma (x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\Sigma (x_i-\bar{x})^2 . \Sigma (y_i-\bar{y})^2}}$$

The six lines from 1150 enable the Student's t value to be found to calculate the 95% confidence limits ± E which depend on the number of readings as well as the standard deviation.

SCX and SCY are the scaling factors for the graph to ensure that all points can be plotted and that whole numbers appear on the axis marker points. If the maximum value is less than 1 then the scale is increased by 10.

Line 2030 plots the axes and the next four loops plot the scale marks. The next two loops PRINT numbers at five division intervals and the following lines print the axis names.

The CHR$(208) is used to enlarge the points on the graph and the best fit line is plotted via line 2090. CHR$(171) is the ± sign needed for the 95% confidence limits which are PRINTed along with the slope, intercept and correlation coefficient

Finally, the X and Y values for the dotted lines are calculated and plotted via SUB 2300.

Sub 2300 is the dotted line routine detailed elsewhere.

```
10 REM Best fit line
20 SYMBOL AFTER 208: H=HIMEM+1: POKE H,240: POKE H+1,240
30 POKE H+16,255: POKE H+18,255: POKE H+23,0: POKE H+22,0
40 INPUT "  Enter the number of pairs of readings (Minimum 3
)";N: IF N<3 THEN PRINT "Not enough readings": GOTO 40
50 DIM E[1,N]
60 PRINT "Type in the names of the variables": PRINT: PRINT
70 PRINT "The name of the abscissa is": PRINT: INPUT A$: PRI
NT
80 PRINT "The name of the ordinate is": PRINT: INPUT B$: CLS
90 FOR P=1 TO N
100 LOCATE 1,P: PRINT "x";P;"=";: INPUT E[0,P]
110 LOCATE 20,P: PRINT "y";P;"=";: INPUT E[1,P]
120 NEXT: REM Or use DATA INPUT (x and y coordinates) to est
ablish N and E[1,N]
130 GOSUB 1000: GOSUB 2000: END
1000 MX=0: MY=0: XM=0: YM=0
1010 FOR P=1 TO N
1020 IF E[0,P]>MX THEN MX=E[0,P]
1030 IF E[1,P]>MY THEN MY=E[1,P]
1040 XM=XM+E[0,P]: YM=YM+E[1,P]
1050 NEXT
1060 XM=XM/N: YM=YM/N
1070 XX=0: XY=0: YY=0
1080 FOR P=1 TO N
1090 XX=XX+(E[0,P]-XM)*(E[0,P]-XM)
1100 XY=XY+(E[0,P]-XM)*(E[1,P]-YM)
1110 YY=YY+(E[1,P]-YM)*(E[1,P]-YM)
1120 NEXT
1130 M=XY/XX: C=YM-M*XM
1140 R=XY/(SQR(XX)*SQR(YY))
1150 DIM T[10]: RESTORE 1150: DATA 12.706,4.103,3.182,2.776,
2.571,2.447,2.365,2.308,2.262,2.228
1160 FOR P=1 TO 10: READ T[P]: NEXT
1170 IF N<=12 THEN T=T[N-2]
1180 IF N>=13 AND N<=27 THEN T=-0.000014*(N-2)^3+0.00152*(N-
2)^2-0.05075*(N-2)+2.5975
1190 IF N>=28 AND N<=62 THEN T=2.056-(N-2)*0.00165
1200 IF N>62 THEN T=1.98
1210 SCX=INT(MX/50)+1: SCY=INT(MY/40)+1
1220 IF MX<=1 THEN SCX=SCX/10
1230 IF MY<=1 THEN SCY=SCY/10
1240 E=T*SQR(1-R*R)*SQR(YY/(N-2))
1250 ERASE T: RETURN
2000 MODE 2: ORIGIN 0,0: CLS
2010 TAG: X=260: Y=385: FOR P=1 TO 13: MOVE X,Y: PRINT MID$(
"BEST FIT LINE",P,1);: X=X+12: NEXT
```

```
2020  X=260: Y=365: FOR P=1 TO 13: MOVE X,Y: PRINT CHR$(210)
;:X=X+12: NEXT
2030 PLOT 600,70: DRAWR -500,0: DRAWR 0,300
2040 FOR P=200 TO 600 STEP 100: PLOT P,70: DRAWR 0,8: NEXT
2050 FOR P=110 TO 600 STEP 10: PLOT P,70: DRAWR 0,5: NEXT
2060 FOR P=145 TO 370 STEP 75: PLOT 100,P: DRAWR  8,0: NEXT
2070 FOR P=70 TO 370 STEP 7.5: PLOT 100,P: DRAWR 5,0: NEXT
2080 TAG
2090 FOR P=1 TO N: MOVE 96+E[0,P]*10/SCX,74+E[1,P]*7.5/SCY:
PRINT CHR$(208);: NEXT
2100 FOR P=200 TO 600 STEP 100: MOVE P-15,62: PRINT (P-100)*
SCX/10;: NEXT
2110 FOR P=145 TO 370 STEP 75: B=1: IF (P-70)*SCY/7.5<=99 TH
EN B=2: IF (P-70)*SCY/7.5<=9 THEN B=3
2120 MOVE 42+10*B,P+8: PRINT (P-70)*SCY/7.5;: NEXT
2130 X=350-5*LEN(A$): Y=38
2140 FOR P=1 TO LEN(A$): Z$=MID$(A$,P,1)
2150 MOVE X,Y: PRINT Z$;
2160 X=X+14+8*(Z$=" "): NEXT
2170 FOR P=1 TO LEN(B$): MOVE 30,230+10*LEN(B$)-20*P: PRINT
MID$(B$,P,1);: NEXT
2180 ORIGIN 100,70
2190 PLOT 0,C*7.5/SCY
2200 DRAWR MX*10/SCX,(MX*M)*7.5/SCY
2210 X1=0: Y1=(E+C)*7.5/SCY: X2=MX*10/SCX: Y2=(MX*M+E+C)*7.5
/SCY
2220 GOSUB 2300
2230 X1=0: Y1=(-E+C)*7.5/SCY: X2=MX*10/SCX: Y2=(MX*M-E+C)*7.
5/SCY
2240 GOSUB 2300:
2250 MOVE 20,300: PRINT "M=";M;: MOVE 350,300: PRINT "C=";C;
2260 MOVE 320,50: PRINT "R=";R;: MOVE 250,25: PRINT "95% Con
f y=";CHR$(32);E;: MOVE 344,25: PRINT CHR$(171);
2270 TAGOFF: ORIGIN 0,0
2280 IF INKEY$="" THEN 2280
2290 MODE 1: RETURN
2300 IF X1=X2 THEN 2370 ELSE IF Y1=Y2 THEN 2330
2310 P=(Y2-Y1)/(X2-X1)
2320 IF ABS(P)<1 THEN 2340 ELSE P=1/P: GOTO 2380
2330 P=0
2340 I=10*SGN(X2-X1)/SQR(1+P*P)
2350 FOR Q=0 TO (X2-X1)/I
2360 PLOT X1+Q*I,Y1+Q*I*P: DRAWR 0.4*I,0.4*I*P: NEXT: RETURN
2370 P=0
2380 I=10*SGN(Y2-Y1)/SQR(1+P*P)
2390 FOR Q=0 TO (Y2-Y1)/I
2400 PLOT X1+Q*I*P,Y1+Q*I: DRAWR 0.4*I*P,0.4*I: NEXT: RETURN
2410 RETURN
```

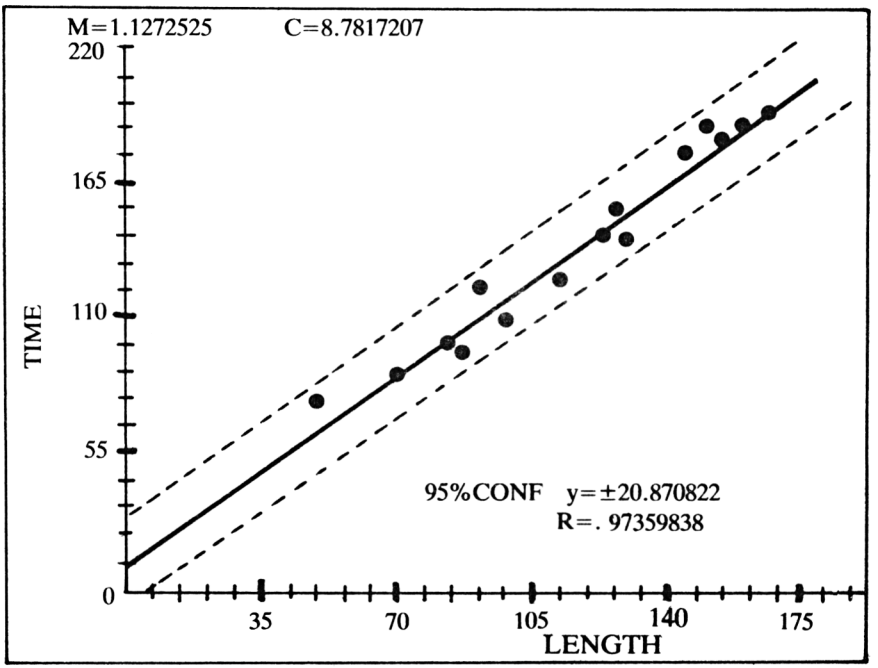


Fig. 3.1 Best Fit Line Display

4. BINARY SEARCH TREE

A Binary Search Tree is a convenient form of storage structure which enables rapid access to the data to be made. For tree terminology see the diagram. A vertex is labelled by A[V], one element of the set. Each vertex U in any subtree to the left has A[U]>A[V] and each vertex in any right subtree has A[U]<A[V]. There is only one vertex for each value in the set. A[V]=O implies an empty vertex.

This subroutine stores N values which must be different and zero is not allowed. By changing A[M] to A$[M], strings can also be handled. (Z→Z$, 0→"" etc.)

Line 50 finds the size of the tree required. If the tree is very long on one subtree, e.g. because the values are put in in order, then it may be necessary to increase the initial value of K. (Subscript out of range message)

As the values are entered, the program goes from the root towards the leaves by finding out

1. if a vertex is already occupied

2. if it is, then is the value less?. If so, go left

3. if not, go right

4. when an empty vertex is found, label it with the value

5. NEXT value

The searching routine follows the same pattern. K1 is the height of the vertex in the tree.

See 'INORDER sequence' for a related subroutine.

```
10 REM Binary Search Tree
20 DEFINT N,K,P,V
30 N=0: WHILE N<=0: INPUT "Number of values";N: WEND
40 M=N+1: K=2
50 WHILE M>1: M=M/2: K=K+1: WEND
60 M=2^K: DIM A[M]: GOSUB 1000: GOSUB 2000: GOTO 1050
1000 INPUT "Median Value";A[1]
1010 FOR P=2 TO N: PRINT "Value";P;: INPUT Z: V=1
```

```
1020 IF Z>A[V] THEN V=V+V+1: IF A[V]<>0 THEN 1020 ELSE A[V]=
Z: GOTO 1040
1030 V=V+V: IF A[V]<>0 THEN 1020 ELSE A[V]=Z
1040 NEXT: RETURN
1050 REM Search Routine
1060 PRINT: INPUT "Is the folowing a member";Z
1070 V=1: K1=1
1080 IF Z=A[V] THEN PRINT Z;" is a member": GOTO 1050
1090 IF K1<K THEN K1=K1+1: IF Z>A[V] THEN V=V+V+1: GOTO 1080
 ELSE V=V+V: GOTO 1080
1100 PRINT Z;" is not a member": GOTO 1050
2000 FOR P=1 TO 2^(K-1)-1: PRINT A[P];SPACE$(7-LEN(STR$(A[P]
)));: NEXT
2010 RETURN
```

EXAMPLE

```
run                          Output
Number of values?  25        52     35     60     20     45
Median value? 52             59     70     10     28     40
Value 2?  35                 48     54      0     65     75
Value 3?  60                  1     15     26      0     37
Value 4?  20                 41      0     49     53     55
Value 5?  45                  0      0     63     69     74
Value 6?  59                  0      0      0      0      0
Value 7?  70                  0      0      0      0      0
Value 8?  10                  0      0      0      0      0
Value 9?  28                  0      0      0      0      0
Value 10? 40                  0      0      0      0      0
Value 11? 48                  0      0      0      0      0
Value 12? 54                  0      0      0
Value 13? 65                 Ready
Value 14? 75
Value 15? 1
Value 16? 15
Value 17? 26
Value 18? 37
Value 19? 41
Value 20? 49
Value 21? 53
Value 22? 55
Value 23? 63
Value 24? 69
Value 25? 74
```

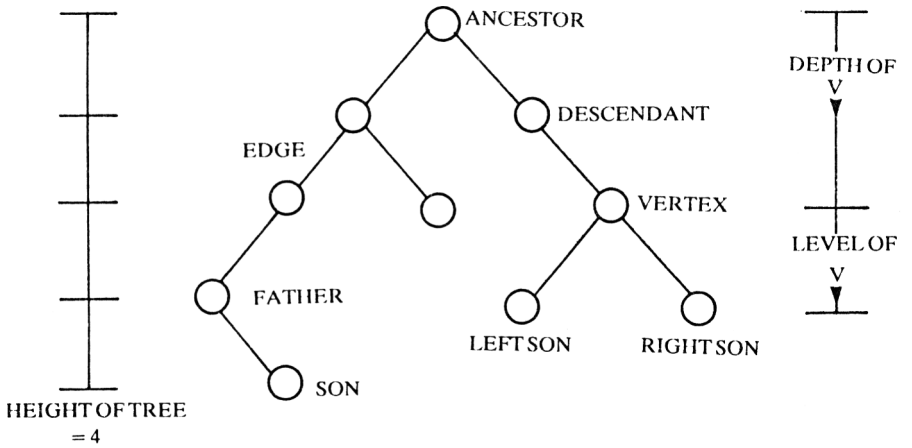


Fig. 4.1(a) Family Type Tree

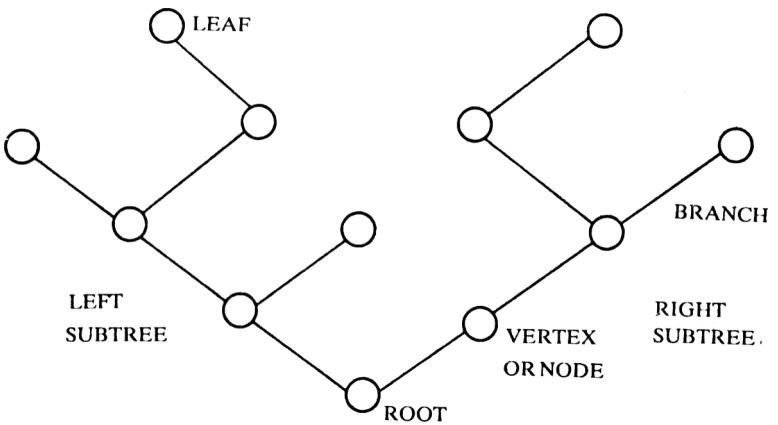


Fig. 4.1(b) Botanical Type Tree

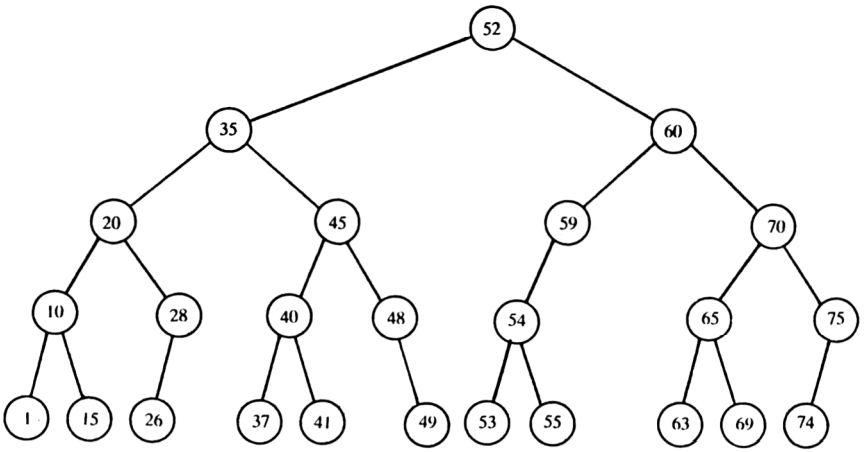


Fig. 4.2 Binary Search Tree

5. BINOMIAL COEFFICIENTS

This subroutine calculates the coefficients of $(a+b)^n$ for $n<=124$

The binomial coefficients are useful in calculating the probability of events happening. For example, choosing 3 balls from a bag containing 3 red and 7 black balls is governed by

$$(3/10+7/10)3 = 1x(.3)^3 +3x(.3)^2 x(.7)+3x(.3)x(.7)^2 +1x(.7)^3$$

$$= 0.027 \quad +0.189 \quad +0.441 \quad +0.343$$

The first term is the probabilty of 3 red balls being picked, the second 2 red and 1 black, the third 1 red and 2 black and finally, the last term in the expansion is all 3 being black.

The illustration is Pascal's triangle, where each row contains a set of binomial coefficients. It has the property that any row can be derived from the row above by adding adjacent coefficients together.

Note the alternative DIM statements using the value of T depending on whether there is a finite or infinite series.

```
10 REM Binomial Coefficients
20 INPUT "Value of n";N: IF ABS(INT(N))<>N THEN PRINT "n is
not a positive whole number. How many terms do you want": IN
PUT T: GOSUB 1010: GOTO 40
30 GOSUB 1000
40 GOSUB 2000: END
1000 DIM B[N+1]: T=N+1: GOTO 1020
1010 DIM B[T]
1020 Z=1: M=N
1030 FOR P=1 TO T: B[P]=Z: Z=Z*M/P: M=M-1: NEXT
1040 RETURN
2000 FOR P=1 TO T: PRINT B[P]: NEXT
2010 ERASE B: RETURN
```

PASCAL'S TRIANGLE

```
10 REM Pascal's Triangle
20 DEFINT H,P,Q
30 H=0: WHILE H<=0: INPUT "Height (up to 9)";H: WEND
40 DIM B[H,H+1]
50 FOR P=0 TO H: Z=1: M=P
60 FOR Q=1 TO P+1: B[P,Q]=Z
70 Z=Z*M/Q: M=M-1
80 NEXT Q,P
90 CLS: PRINT: PRINT
100 FOR P=0 TO H: PRINT SPACE$(18-P*2);
110 FOR Q=1 TO P+1: A$=STR$(B[P,Q])
120 PRINT SPACE$(4-LEN(A$));A$;: NEXT Q: PRINT
130 PRINT: NEXT P
```

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5  10  10   5   1
        1   6  15  20  15   6   1
      1   7  21  35  35  21   7   1
    1   8  28  56  70  56  28   8   1
  1   9  36  84 126 126  84  36   9   1
```

6. CIRCLE

These subroutines have been used for calculating the positions of circle centres from engineering drawings and for approximating complex shapes to a series of arcs for NC machine operation.

The first calculates the centre coordinates and the radius of the only circle that passes through three points. Note that as there are two values to a square root, R may be positive or negative. Care is therefore needed if you use it as an absolute value in a formula.

The second subroutine calculates the position of the centres of the two circles of a given diameter which pass through two points.

```
10 REM Finding the centre of the circle
20 CLS: PRINT "   Insert the coordinates of the three  point
s": PRINT
30 INPUT "x1=";X1: LOCATE 20,4: INPUT "y1=";Y1
40 INPUT "x2=";X2: LOCATE 20,5: INPUT "y2=";Y2
50 INPUT "x3=";X3: LOCATE 20,6: INPUT "y3=";Y3
60 GOSUB 1000: GOSUB 2000: END
1000 D=X1*(Y2-Y3)+X2*(Y3-Y1)+X3*(Y1-Y2): IF ABS(D)<0.000001
THEN GOSUB 2050: GOTO 20
1010 OX=((X1*X1+Y1*Y1)*(Y2-Y3)+(X2*X2+Y2*Y2)*(Y3-Y1)+(X3*X3+
Y3*Y3)*(Y1-Y2))/2/D
1020 OY=((X1*X1+Y1*Y1)*(X3-X2)+(X2*X2+Y2*Y2)*(X1-X3)+(X3*X3+
Y3*Y3)*(X2-X1))/2/D
1030 R=SQR((OX-X1)*(OX-X1)+(OY-Y1)*(OY-Y1))
1040 RETURN
2000 PRINT: PRINT "Radius=";R
2010 PRINT: PRINT "Coordinates of the centre are"
2020 PRINT: PRINT "Ox=";OX
2030 PRINT: PRINT "Oy=";OY
2040 RETURN
2050 PRINT "    Error in the data. Press any key to begin ag
ain"
2060 IF INKEY$="" THEN 2060
2070 RETURN
```

```
10 REM Finding the centres of TWO circles
20 CLS: PRINT "Insert the coordinates of the two points": PR
INT
30 INPUT "x1=";X1: LOCATE 20,4: INPUT "y1=";Y1: PRINT
40 INPUT "x2=";X2: LOCATE 20,6: INPUT "y2=";Y2: PRINT
50 PRINT "Insert the radius of the circles": PRINT
60 INPUT "r=";R
```

```
70 GOSUB 1000: END
1000 DEF FN R(A)=A*-(ABS(A)>0.000001): NL$=CHR$(10)+CHR$(13)
1010 IF (X1-X2)*(X1-X2)+(Y1-Y2)*(Y1-Y2)>4*R*R THEN PRINT "Er
ror in the data.      Press any key to re-enter": GOSUB 1120:
 RUN
1020 IF ABS(X2-X1)<0.000001 AND ABS(Y2-Y1)<0.000001 THEN PRI
NT "Arc too small to calculate centres.      Press any key":
GOSUB 1120: RUN
1030 IF ABS(Y2-Y1)<0.000001 THEN THETA=PI/2: GOTO 1050
1040 THETA=ATN((X1-X2)/(Y1-Y2))
1050 XM=(X1+X2)/2: YM=(Y1+Y2)/2
1060 P=SQR(R*R-(Y1-YM)*(Y1-YM)-(X1-XM)*(X1-XM))
1070 PC=P*COS(THETA): PS=P*SIN(THETA)
1080 PRINT: PRINT
1090 PRINT "Centres are at";NL$;FN R(XM+PC);",";FN R(YM+PS)
1100 PRINT "and";NL$;FN R(XM-PC);",";FN R(YM-PS)
1110 RETURN
1120 IF INKEY$="" THEN 1120
1130 RETURN
```

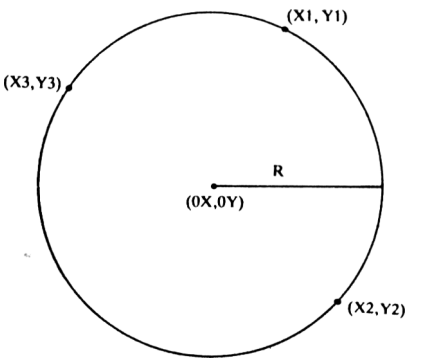


Fig. 6.1 Circle Through Three Points

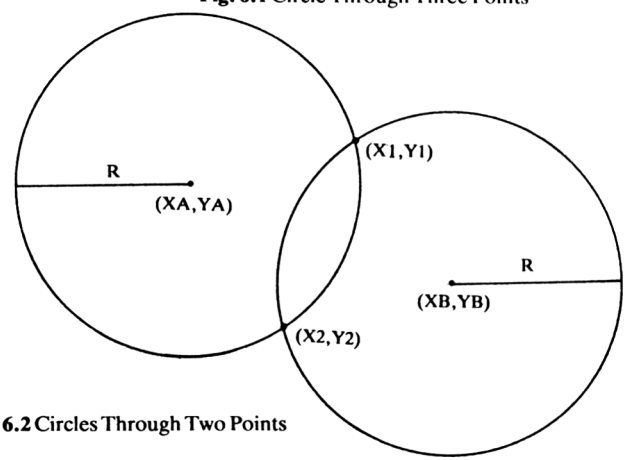


Fig. 6.2 Circles Through Two Points

7. COMBINATIONS OF PLUS AND MINUS ONE IN GROUPS OF THREE

This subroutine was developed for the cubic crystal program and produces the eight combinations of +1 and −1 in groups of three leaving the input unchanged at the end. Each choice can be made in two ways, +1 or −1, and so the total number of different arrangements is 2×2×2=8.

The first arrangement is

1 1 1

Two numbers are interchanged or one is multiplied by −1 giving a sequence of changes as follows:

A=0	1	1	1	A(2)*−1
1	1	1	−1	A(1)⟷A(2)
2	1	−1	1	A(2)*−1
3	1	−1	−1	A(0)*−1
4	−1	−1	−1	A(2)*−1
5	−1	−1	1	A(1)⟷A(2)
6	−1	1	−1	A(2)*−1
7	−1	1	1	A(0)*−1
8	1	1	1	

```
10 REM Combinations of plus and minus one in groups of three
20 GOSUB 1000: GOSUB 2000: END
1000 DIM A[2],C[8,2]
1010 A[0]=1: A[1]=1: A[2]=1
1020 FOR A=0 TO 8: FOR B=0 TO 2
1030 C[A,B]=A[B]: NEXT
1040 IF A=1 OR A=5 THEN Z=A[1]: A[1]=A[2]:A[2]=Z: GOTO 1070
1050 IF A=3 OR A=7 THEN A[0]=-A[0]: GOTO 1070
1060 A[2]=-A[2]
1070 NEXT
1080 RETURN
2000 FOR A=0 TO 8: FOR B=0 TO 2
2010 PRINT C[A,B];: NEXT
2020 PRINT: NEXT:
2030 ERASE A,C
2040 RETURN
```

8. COMPLEX NUMBERS

Complex numbers of the form a+bi (where i is the square root of -1) are a very convenient way of representing pairs of numbers. In geometrical terms, a is the x−coordinate and b the y−coordinate and complex numbers occur quite frequently in mathematics.

At first sight, it might appear difficult to handle complex numbers in a computer as SQR(−1) is not recognised by the computer. However, by keeping the real and imaginary parts of a complex number separate, say in two arrays, they are no more difficult to deal with than ordinary numbers. It is necessary to use a string PRINT to incorporate the "i" in the final answer.

These subroutines deal with the simple operations of addition, subtraction, multiplication and division but more complicated operations can be found under 'MATRICES'.

The formulae used are as follows;−

(a+bi) plus (c+di)=(a+c)+(b+d)i
(a+bi) minus (c+di)=(a−c)+(b−d)i
(a+bi) times (c+di)=(ac−bd)+(bc+ad)i
(a+bi) divided by (c+di)=(ac+bd)/(c^2 +d^2)+(bc−ad)/(c^2 +d^2)i

The DEF FN for the final PRINT is to enable the normal mathematical conventions to be observed and avoid clumsy expressions such as

0±2i i.e. −2i and −2+0i i.e. −2

```
10 REM Complex numbers
20 DEF FN A$(A$,A,BOOLE)=MID$(A$,1+A,-LEN(A$)*BOOLE)
30 PRINT "1st Complex Number Z1=a+ib"
40 PRINT: INPUT "a=";A: INPUT "b=";B
50 PRINT "2nd Complex Number Z2=c+id"
60 PRINT: INPUT "c=";C: INPUT "d=";D
70 CLS: GOSUB 2000: END
2000 R=A: I=B: GOSUB 2100: PRINT: PRINT "Z1=";Z$
2010 R=C: I=D: GOSUB 2100: PRINT: PRINT "Z2=";Z$
2020 R=A+C: I=B+D: GOSUB 2100: PRINT: PRINT "Z1+Z2=";Z$
2030 R=A-C: I=B-D: GOSUB 2100: PRINT: PRINT "Z1-Z2=";Z$
2040 R=A*C-B*D: I=B*C+A*D: GOSUB 2100: PRINT: PRINT "Z1*Z2="
;Z$
2050 S=C*C+D*D: IF S=0 THEN GOTO 2080
2060 R=(A*C+B*D)/S: I=(B*C-A*D)/S: GOSUB 2100: PRINT: PRINT
"Z1/Z2="Z$
```

```
2070 PRINT: RETURN
2080 PRINT: PRINT "Denominator zero, infinite answer for div
ision"
2090 PRINT: RETURN
2100 Z$=FN A$("-",0,R<0)+FN A$(STR$(R),1,R<>0)+FN A$("+",0,A
BS(R)>0 AND I>0)+FN A$("-",0,I<0)+FN A$(STR$(I),1,I<>0 AND A
BS(I)<>1)+FN A$("i",0,I<>0)+FN A$("0",0,R=0 AND I=0)
2110 RETURN
```

EXAMPLE

```
run
1st complex number Z1=a+bi

a=? 3
b=? -1

2nd complex number Z2=c+di

c=? -3
d=? 4

Z1=3-i
Z2=-3+4i
Z1+Z2=3i
Z1-Z2=6-5i
Z1*Z2=-5+15i
Z1/Z2=-.52-.36i

Ready
```

9. COMPREHENSIVE NUMBER FILTER

This filter subroutine only allows positive and negative integers, decimals or exponential numbers through. The number is entered as a string and tested character by character.

Line 1000 initialises the variables. Q is the position of the decimal point in the mantissa and R the position of "E" or "e" if they are present. S is the position after the exponent of any "+" or "−". W is used to check that there is a character after this "+" or "−" position. FAIL is used for the RETURN from the tests. All these variables are set to zero

T is initially set to 1 but changes to 2 if "+" or "−" is used in front of the number so that they are not included in the tests.

Line 1040 looks for an exponential form of number and continues the tests at 1110.

Line 1050 tests for a decimal in non−exponential numbers and line 1060 rejects entries such as "+.", "+", "." and those >10 ↑ 38 or <10 ↑ −38.

Line 1100 fails exponential numbers with no mantissa. The following lines search for a decimal point in the mantissa and reject exponential numbers with no exponent or with a decimal point in the exponent. The position of any "+" or "−" after the exponent is found and numbers with no entries after the sign are rejected.

Returning to 1110, all characters except the initial "+" or "−", the "E" or "e" the decimal point and the exponent "+" or "−" are tested for numerical characteristics.

Finally, the numerical value of an exponent number is tested to see that it lies between 10 ↑ 38 and 10 ↑ −38 having first dealt with any number with a mantissa equal to zero (because of the LOG required in the test).

```
10 REM Comprehensive Number Filter
20 INPUT "Number";Q$: GOSUB 1000
30 IF FAIL THEN 20 ELSE GOSUB 2000: END
1000 Q=0: R=0: S=0: FAIL=0: T=1: L=LEN(Q$)
1010 IF Q$="" THEN GOSUB 1170: RETURN
1020 IF LEFT$(Q$,1)="-" OR LEFT$(Q$,1)="+" THEN T=2
1030 FOR P=T TO L: Z$=MID$(Q$,P,1)
1040 IF Z$="E" OR Z$="e" THEN 1110 ELSE NEXT P
1050 FOR P=T TO L: IF MID$(Q$,P,1)="." THEN Q=P
```

```
1060 NEXT P: IF Q=T AND L>38 OR L-Q>38 OR L<T OR L<Q+1 THEN
GOSUB 1170: RETURN
1070 FOR P=T TO L: IF P<>Q AND P<>R AND P<>S THEN IF MID$(Q$
,P,1)<"0" OR MID$(Q$,P,1)>"9" THEN GOSUB 1170: RETURN
1080 NEXT P: IF R<>0 THEN GOSUB 1180
1090 RETURN
1100 IF P<=T THEN GOSUB 1170: RETURN
1110 R=P: FOR P=T TO R-1: IF MID$(Q$,P,1)="." THEN Q=P
1120 NEXT P: IF L=R OR R<=Q+1 THEN GOSUB 1170: RETURN
1130 IF MID$(Q$,R+1,1)="+" OR MID$(Q$,R+1,1)="-" THEN S=R+1:
 W=1
1140 IF L<=R+W THEN GOSUB 1170: RETURN
1150 GOTO 1070
1170 FAIL=-1: RETURN
1180 IF VAL(MID$(Q$,T,R-T))=0 THEN Q$="0": RETURN
1190 V=VAL(MID$(Q$,R+1))+LOG10(VAL(MID$(Q$,T,R-T))): IF ABS(
V)>=38 THEN GOSUB 1170: RETURN
1200 RETURN
2000 N=VAL(Q$): PRINT N: RETURN
```

10. CONDITIONAL BRACKETS

This subroutine puts brackets around certain items when PRINTed according to the outcome of a logical operation. For example in accounting brackets are sometimes used to indicate a negative entry rather than red ink or a minus sign. The variable BOOLE would then be (N<0).

The routine works by changing the code of the character string from 32 (blank) to 40 or 41, the left hand and right hand brackets respectively. This technique can be used for other characters and is used in 'DETERMINANT BY LAPLACE DEVELOPMENT' to change "+" into "−" in the string expression.

```
10 REM Conditional Brackets
20 DEF FN L$(BOOLE)=CHR$(32-8*BOOLE)
30 DEF FN R$(BOOLE)=CHR$(32-9*BOOLE)
35 REM The variable BOOLE can only take the values 0(False)
or -1(True)
40 DEF FN P$(A)=MID$(STR$(A),2)
50 GOSUB 1000: END
1000 FOR A=1 TO 4: FOR B=1 TO 5
1010 PRINT FN L$(A<B);FN P$(A-B);FN R$(A<B),A-B
1020 NEXT B,A: RETURN
```

11–16. CONVERSIONS

Conversions between binary, decimal and hexadecimal are available in Amstrad but are limited to sixteen bits. The following routines can be used for positive integers of any size.

11. BINARY TO DECIMAL

This routine finds the decimal number by adding the number of 1's, 2's, 4's etc..

```
10 REM Binary to Decimal
20 REM Use Binary Number test routine
30 INPUT "Input binary number";B$
40 GOSUB 1000: GOSUB 2000: END
1000 D=0: C=1: L=LEN(B$)
1010 D=D+2^(L-C)*VAL(MID$(B$,C,1))
1020 IF L=C THEN RETURN ELSE C=C+1: GOTO 1010
1030 RETURN
2000 PRINT B$;SPC(5);D: RETURN
```

12. BINARY TO HEXADECIMAL

There is a simple relationship between binary and hexadecimal numbers as each group of four in a binary number represents a hex. digit. The Function B$(S) is the list of 16 hex. digits from which the correct one can be selected by S which is formed by four cycles of Q and terminated when T=L

```
10 REM Binary to Hexadecimal
20 DEF FN B$(S)=MID$("0123456789ABCDEF",S,1)
30 LINE INPUT "Binary Number ";B$
40 REM Use Binary Number test routine
50 GOSUB 1000: GOSUB 2000: END
1000 L=LEN(B$): H$=STRING$(1+INT((L-1)/4),32): T=0
1010 S=0: Q=0
1020 S=S+2^Q*VAL(MID$(B$,L-Q-INT(T/4)*4,1)): Q=Q+1: T=T+1
1030 IF T=L THEN MID$(H$,1,1)=FN B$(S+1): RETURN
1040 IF Q=4 THEN MID$(H$,LEN(H$)-T/4+1)=FN B$(S+1): GOTO 101
0 ELSE GOTO 1020
2000 PRINT B$;SPC(5);H$
2010 RETURN
```

13. DECIMAL TO BINARY

This routine is based on successive division by 2 and finding the remainder. As there are rounding errors in the computer calculations, 0.0000001 is added to give the correct INT value.

```
10 REM Decimal to Binary
20 DEFINT P,Q
30 INPUT "Input decimal number";D
40 GOSUB 1000: GOSUB 2000: END
1000 P=1: WHILE 2^P<ABS(D)+1: P=P+1: WEND
1010 A$=STRING$(P,32): FOR Q=1 TO P
1020 U=INT(D/2^Q-INT(D/2^Q+0.0000001)+0.5)
1030 MID$(A$,1+P-Q)=RIGHT$(STR$(U),1): NEXT
1040 RETURN
2000 PRINT D;SPC(5);A$
2010 RETURN
```

14. DECIMAL TO HEXADECIMAL

This routine slices the hexadecimal list in FN H$(H) with the remainders from division by powers of 16.

```
10 REM Decimal to Hexadecimal
20 DEF FN H$(H)=MID$("0123456789ABCDEF",1+16*(H/16-INT(H/16)
),1)
30 INPUT "Input decimal number";N: IF N<0 THEN 30
40 GOSUB 1000: GOSUB 2000: END
1000 IF N=0 THEN A$="0": RETURN
1010 A$=""
1020 FOR D=LEN(STR$(N))-2 TO 0 STEP -1
1030 A$=A$+FN H$(INT(N/(16^D))): NEXT
1040 IF LEFT$(A$,1)="0" THEN A$=RIGHT$(A$,LEN(A$)-1): GOTO 1
040
1050 RETURN
2000 PRINT N;SPC(5);A$
2010 RETURN
```

15A. HEXADECIMAL TO BINARY 1.

The Function H(A) converts H$ into a number by using the fact that the digits are 48–57 and the capitals A–F are 65–70 in the ASCII list of Codes. For example, ASC(C) is 67, take off 48 and 7 ("C">"A" hence the bracket equals –7) which gives 12–the hex. value of C.

```
10 REM Hexadecimal to Binary 1.
20 DEF FN H(A)=ASC(MID$(H$,A,1))-48+7*(MID$(H$,A,1)>="A")
30 LINE INPUT "HEX. NUMBER ";H$
40 GOSUB 1000: GOSUB 2000: END
1000 B$=STRING$(LEN(H$)*4,32)
1010 FOR N=LEN(H$) TO 1 STEP -1: FOR M=1 TO 4
1020 U=INT(FN H(N)/2^M-INT(FN H(N)/2^M)+0.5000005)
1030 MID$(B$,4*N-M+1)=RIGHT$(STR$(U),1): NEXT M,N
1040 RETURN
2000 FOR T=0 TO 2: IF MID$(B$,1+T,1)="0" THEN NEXT T
2010 PRINT H$;SPC(5);RIGHT$(B$,LEN(B$)-T)
2020 RETURN
```

15B. HEXADECIMAL TO BINARY 2.

The Function H(A) unscrambles the hexadecimal number using the order of letters and numbers in the ASCII list and slicing with ASC(MID$(H$,A,1))–47. For example, ASC(F)=70 so that the 23rd character of the string is selected i.e. the "?" which has an ASC value of 63. Take off 48 and this gives 15–the decimal value of F.

```
10 REM Hexadecimal to Binary 2. Upper or lower case
20 DEF FN H(A)=ASC(MID$("0123456789*******:;<=>?***********
**************:;<=>?",ASC(MID$(H$,A,1))-47,1))-48
30 LINE INPUT "H$ (UPPER or lower case) ";H$
40 GOSUB 1000: GOSUB 2000: END
1000 B$=STRING$(LEN(H$)*4,32)
1010 FOR N=LEN(H$) TO 1 STEP -1: FOR M=1 TO 4
1020 U=INT(FN H(N)/2^M-INT(FN H(N)/2^M)+0.5000005)
1030 MID$(B$,4*N-M+1)=RIGHT$(STR$(U),1): NEXT M,N
1040 RETURN
2000 FOR T=0 TO 2: IF MID$(B$,T+1,1)="0" THEN NEXT T
2010 PRINT H$;SPC(5);RIGHT$(B$,LEN(B$)-T)
2020 RETURN
```

Successive division produces U which is built up in B$ to form the binary number. The loop in T is to remove surplus zeros from the front of the binary number.

16A. HEXADECIMAL TO DECIMAL 1.

The Function H(A) converts H$ as in the Hex. to Binary routine. It only works for upper case hence the UPPER$ in line 30.

```
10 REM Hexadecimal to Decimal 1.
20 DEF FN H(A)=ASC(MID$(H$,A,1))-48+7*(MID$(H$,A,1)>="A")
30 LINE INPUT "H$ ";H$: H$=UPPER$(H$)
40 GOSUB 1000: GOSUB 2000: END
1000 D=0: FOR N=1 TO LEN(H$)
1010 D=D+FN H(N)*16^(LEN(H$)-N): NEXT: RETURN
2000 PRINT H$;SPC(5);D
2010 RETURN
```

16B. HEXADECIMAL TO DECIMAL 2.

The more complicated Function is used as before to deal with small and capital letters in the hexadecimal number.

```
10 REM Hexadecimal to Decimal 2.
20 DEF FN H(A)=ASC(MID$("0123456789*******:;<=>?***********
**************:;<=>?",ASC(MID$(H$,A,1))-47,1))-48
30 LINE INPUT "Hex. number, UPPER or lower case ";H$
40 GOSUB 1000: GOSUB 2000: END
1000 D=0: FOR N=1 TO LEN(H$)
1010 D=D+FN H(N)*16^(LEN(H$)-N): NEXT
1020 RETURN
2000 PRINT H$;SPC(5);D
2010 RETURN
```

17–20.CHECKING DATA INPUT

The purpose of a data input routine is twofold. Firstly, it should enable the input to be checked automatically, to ensure that it is the right type and within range i.e. a number, a name, single letters etc and secondly, it should enable it to be checked by the keyboard operator to see that the values are correct, the spelling right etc. before it is used. Unchecked data will inevitably lead to the program crashing at some time. For example INPUT N: N=1/N will lead to a 'Division by Zero'if the "<ENTER>" key is accidentally pressed before the digit key. This would simply be avoided by A INPUT N: IF N=0 THEN A

Other inclusive tests trap this error. For example IF MID$(N$,P,1)>="0" AND MID$(N$,P,1)<="9"(In a loop) THEN accept ensures all the characters in a number are digits. If you are setting up a DIM statement you must ensure that it cannot be negative ie. IF VAL(N$)<0 THEN(INPUT Line No.)

A generalised form of INPUT is

Line No. LET Q =(Line No.): INPUT Q$

Q$ can then be tested character by character with a loop and if it fails THEN GOTO Q. If it passes Q$ is then either converted to a number via a VAL function and stored or stored directly leaving Q and Q$ available for the next INPUT.

17. DATA INPUT (Linear equations with up to eight variables)

This subroutine accepts coefficients and constants for sets of equations (maximum number of 8) prior to solving them as simultaneous equations, for example.

The limitation of 8 arises only from the size of the screen. The FN BS$ is for erasing and back spacing and the FN A$ is for printing in packed format. FN K and FN J avoid complications in the PRINT locations.

The first half of the routine PRINTs the equations in algebraic form by concatenating CHR$ (96+K) with FN A$(J),"*" and CHR$(90+K−N) to form a1*x,b2*y etc.. At 1090 the numerical values are inserted and a1 is overwritten by a blank, the PRINT position is backspaced and E[J,K] written where a1*x was before.

If you are not satisfied with the numerical values that have been inserted, you can go back to 1090 and change them via K$.

```
10 REM DATA INPUT (Linear Equations)
20 SYMBOL AFTER 208: POKE HIMEM+1,0
30 DEFINT J,K,N
40 DEF FN BS$(A)=SPACE$(A)+STRING$(A,8)
50 DEF FN A$(A)=MID$(STR$(A),2)
60 DEF FN K(K)=8*K+32*(K>4)-6: DEF FN J(J)=2*J-(K>4)+4
70 INPUT "How many variables are there";N: IF N<1 OR N>8 THE
N 70
75 DIM E[N,N+1]
80 GOSUB 1000: END
1000 CLS: LOCATE 10,2: PRINT "Your equations are"
1010 LOCATE 10,3: FOR P=1 TO 18: PRINT CHR$(208);: NEXT
1020 FOR J=1 TO N: FOR K=1 TO N
1030 LOCATE FN K(K),FN J(J): PRINT CHR$(96+K)+FN A$(J)+"*"+C
HR$(90+K-N): NEXT
1040 FOR K=1 TO N-1:
1050 LOCATE FN K(K)+5,FN J(J): PRINT "+": NEXT
1060 LOCATE 30,FN J(J): PRINT " = "+"K"+FN A$(J): NEXT
1070 FOR K=1 TO N
1080 LOCATE 33,4: PRINT "Const": LOCATE FN K(K)-2*(K>4),4: P
RINT CHR$(32-15*(K>4))+CHR$(90+K-N): NEXT
1090 FOR J=1 TO N: FOR K=1 TO N+1
1100 LOCATE 1,22
1110 PRINT "Now enter the values ";CHR$(96+K+(K=N+1)*(K+21))
+FN A$(J)+"=";SPC(9);:LOCATE 24,22: INPUT E[J,K]
1120 LOCATE (1-FN K(K))*(K<>N+1)-(K=N+1)*32,2*J+4+(K>4)*(N>4
): PRINT USING "&"; FN BS$(7)+STR$(E[J,K])
1130 NEXT K,J
1140 LOCATE 1,22: PRINT SPACE$(40)
1150 LOCATE 1,22: PRINT "   OK?   (Y/N)";: INPUT K$
1160 K$=UPPER$(K$): IF K$="Y" THEN RETURN ELSE 1090
```

EXAMPLE

```
N=5
                Your Equations are

V/Z    W       X       Y        Const
a1*v + b1*w + c1*x + d1*y +
e1*z                             =k1
a2*v + b2*w + c2*x + d2*y +
e2*z                             =k2
a3*v + b3*w + c3*x + d3*y +
e3*z                             =k3
a4*v + b4*w + c4*x + d4*y +
e4*z                             =k4
```

```
a5*v + b5*w + c5*x + d5*y +
e5*z                          =k5

Now insert the values a1=?

V/Z     W       X       Y       Const
15    + 25    + 9.5   ± 5     +
7                               =27
9.6   ± 1Ø    + 4.3   + 7     +
2                               =51
3.7   + 16    + 6.9   + 13    +
4                               =97
16    + 91    + 2     + 81    +
6                               =115
5     + 2     + 15    + 6     +
8                               =23

ok? (y/n)
```

18. DATA INPUT (Matrices or arrays)

This subroutine enables a matrix or array of numbers to be entered and checked. The initial display is in algebraic format. FN A$(A) is used for printing in a packed form. At 1030 the numerical values are inserted using the erase and backspace function BS$(A). If an error is made, the values can be changed by going back to 1030.

```
10 REM Data Input (Matrices or arrays)
20 DEFINT J,K,M,N
30 DEF FN A$(A)=MID$(STR$(A),2)
40 DEF FN BS$(A)=SPACE$(A)+STRING$(A,8)
50 CLS: PRINT: PRINT " Insert the size of the matrix as m ro
ws and n columns"
60 LOCATE 1,5: INPUT "m=";M: LOCATE 20,5: INPUT "n=";N: IF N
=0 OR M=0 THEN 50
70 CLS: PRINT: PRINT "     The matrix is"
80 DIM A[M,N]: K$="Y"
85 GOSUB 1000: GOSUB 2000: END
1000 FOR J=1 TO M: FOR K=1 TO N
1010 LOCATE 5*K-3,2*J+3: PRINT "a"+FN A$(J)+FN A$(K)
1020 NEXT K,J
1030 FOR J=1 TO M: FOR K=1 TO N
1040 LOCATE 1,20: PRINT "   Now type in the numerical values
";
1050 IF K$<>"Y" THEN PRINT " again, correctly"
1060 PRINT: PRINT "a"+FN A$(J)+FN A$(K)+" is ";FN BS$(8);: I
NPUT A[J,K]
1070 LOCATE 5*K-3,2*J+3: PRINT FN BS$(5);A[J,K]
1080 NEXT K,J
1090 LOCATE 1,20: PRINT FN BS$(80);"      OK?  (Y/N)";: INPUT
 K$: K$=UPPER$(K$)
1100 IF K$<>"Y" THEN 1030
1110 RETURN
2000 CLS: TAG: MOVE 124,395: PRINT "T H E  M A T R I X  I S"
;
2010 PLOT 124,378: DRAWR 368,0
2020 FOR J=1 TO M: FOR K=1 TO N
2030 MOVE 80*K-40,400-40*J: PRINT A[J,K];: NEXT K,J
2040 MOVE 72,368: DRAWR -40,0: DRAWR 0,-32-40*(M-1): DRAWR 4
0,0
2050 MOVE 80*N+40,368: DRAWR 40,0: DRAWR 0,-32-40*(M-1): DRA
WR -40,0
2060 TAGOFF: RETURN
```

EXAMPLE

```
Insert the size of the matrix as m rows and n columns

m=? 5          n=? 4

         The matrix is

a11    a12    a13    a14
a21    a22    a23    a24
a31    a32    a33    a34
a41    a42    a43    a44
a51    a52    a53    a54

           Now type in the numerical values

         a11 is?  1

 1      3      5      7
 2      4      6      8
-1      4      6      2.5
 3      9      11     17
 6      5      3      1

        OK? (y/n)   y

            THE MATRIX IS

⎡ 1      3      5      7   ⎤
⎢ 2      4      6      8   ⎥
⎢-1      4      6      2.5 ⎥
⎢ 3      9      11     17  ⎥
⎣ 6      5      3      1   ⎦
```

19. DATA INPUT (Single Variable)

This subroutine is for entering data for subsequent statistical analysis.

With single variables, you often have too many data to display them all at once. In this routine, they are stored in V[N]. However each is checked for numerical characteristics in SUB 1120 before storing. The use of INKEY$ enables the number to be built up character by character before testing and this avoids the complications of INPUT A$ and a comma.

After the numbers are all tested, they are displayed in a column using VPOS(#0) to stop the flow and allow the numbers to be checked 20 at a time.

If an alteration is required (line 1250), the variable subscript number is sought and checked (line 1330) before the new value is asked for (lines 1340–1360).

The revised 20 values are again displayed for further possible correction. If they are OK then the routine displays the next 20 and so on.

```
10 REM DATA INPUT (Single Variable)
20 DEFINT A,K,N,P,Q,S,T
30 DEF FN A$(A)=MID$(STR$(A),2)
40 A$=STRING$(10,32)+STRING$(10,8): NL$=CHR$(10)+CHR$(13)
50 CLS: PRINT: INPUT "How many variables are there";N
60 IF N>=4 THEN 65 ELSE PRINT STR$(N)+" values are not enoug
h to do a sensible statistical analysis.": PRINT NL$;"Press
any key to begin again.": GOSUB 1100: CLS: GOTO 50
65 GOSUB 1000: GOSUB 2000: END
1000 DIM V[N]: CLS
1010 WINDOW #2,1,40,22,25
1020 PRINT #2,"Insert the values"
1030 FOR P=1 TO N: V$=""
1040 LOCATE #2,1,2: PRINT #2,"   V(";FN A$(P);")  is ";: PRI
NT #2,A$;
1050 T$=INKEY$: IF T$="" THEN 1050
1060 PRINT #2,T$;: IF T$=CHR$(13) THEN 1070 ELSE V$=V$+T$: G
OTO 1050
1070 GOSUB 1120: IF FAIL THEN P=P-1 ELSE V[P]=VAL(V$)
1080 NEXT
1090 CLS :GOSUB 1170: RETURN
1100 IF INKEY$="" THEN 1100
1110 RETURN
1120 FAIL=0: S=0: T=1: IF LEFT$(V$,1)="+" OR LEFT$(V$,1)="-"
 THEN T=2
1130 FOR Q=T TO LEN(V$): Z$=MID$(V$,Q,1)
1140 IF Z$="." THEN S=S+1
1150 IF Z$="." OR Z$>="0" AND Z$<="9" THEN NEXT: IF S<=1 THE
N RETURN
1160 FAIL=-1: RETURN
```

```
1170 A=1
1180 PRINT: PRINT #2, "Are these values correct?": FOR P=A T
O N
1190 PRINT TAB(10);"V(";FN A$(P);")=";V[P]
1200 IF VPOS(#0)>20 THEN 1220
1210 NEXT
1220 GOTO 1240
1225 IF P>N THEN RETURN
1230 CLS : GOTO 1190
1240 PRINT #2, "(Y/N)";
1250 INPUT #2,K$: K$=UPPER$(K$)
1260 IF K$="Y" THEN A=P+1: GOTO 1225
1270 CLS #2: PRINT #2,"Which entry would you like to alter?"
;
1280 V$=""
1290 T$=INKEY$: IF T$="" THEN 1290
1300 PRINT #2,T$;: IF T$=CHR$(13) THEN 1310 ELSE V$=V$+T$: G
OTO 1290
1310 GOSUB 1120: IF FAIL THEN 1270
1320 IF LEFT$(V$,1)<>"-" THEN K=VAL(V$) ELSE 1270
1330 IF K>N OR K>A+18 OR K<A OR K<=0 THEN PRINT "Not valid":
 GOTO 1270
1340 CLS #2: V$="": PRINT #2,"Please enter the correct value
 V(";FN A$(ABS(K));")=";
1350 T$=INKEY$: IF T$="" THEN 1350
1360 PRINT #2,T$;: IF T$=CHR$(13) THEN 1370 ELSE V$=V$+T$: G
OTO 1350
1370 GOSUB 1120: IF FAIL THEN PRINT "Not a valid entry": GOT
O 1340
1380 V[K]=VAL(V$): CLS: GOTO 1180
2000 CLS: PRINT: FOR P=1 TO N
2010 PRINT TAB(10);"V[";FN A$(P);"]=";V[P]: NEXT
2020 RETURN
```

20. DATA INPUT (x and y Coordinates, Statistical Data etc.)

This subroutine accepts pairs of values, allows for checking and correction and then stores them.

The problem of having a lot of data is overcome by dealing with them in groups of ten. Array B is chosen to be a multiple of ten and may be larger than is just necessary to hold the data. The numbers are input using INKEY$ for each digit and when complete each number is tested in SUB 1400 for numerical characteristics.

When the input operation is completed, the numbers are displayed in groups of 10 and can be altered if necessary. Each group can be displayed after changing until you are satisfied. R is used as the block counter.

When you are satisfied with the data, they are transferred to array E which is the correct size.

```
10 REM DATA INPUT (x and y coordinates)
20 DEFINT L,N,P,Q,R,S: DEF FN A$(A)=MID$(STR$(A),2)
30 BS$=STRING$(10,32)+STRING$(10,8)
40 CLS: LOCATE 2,2: INPUT "Number of pairs of readings";N: I
F N<=0 THEN 40
50 DIM E[1,N]: GOSUB 1000: END
1000 DIM B[1,10*INT((N-1)/10+1)]: CLS
1010 WINDOW #1,1,40,18,21: WINDOW #2,1,40,22,25
1020 LOCATE #1,1,1: PRINT #1,"   Type in the values of the x
 and y        coordinates"
1030 FOR P=1 TO N: P$=FN A$(P)
1040 LOCATE #2,5,1: PRINT #2,BS$;"x";P$;"=";: Z$=""
1050 T$=INKEY$: IF T$="" THEN 1050
1060 PRINT #2,T$;
1070 IF T$<>CHR$(13) THEN Z$=Z$+T$: GOTO 1050
1080 GOSUB 1400: IF FAIL THEN 1040 ELSE B[0,P]=VAL(Z$)
1090 LOCATE #2,5,3: PRINT #2,BS$;"y";P$;"=";: Z$=""
1100 T$=INKEY$: IF T$="" THEN 1100
1110 PRINT #2,T$;
1120 IF T$<>CHR$(13) THEN Z$=Z$+T$: GOTO 1100
1130 GOSUB 1400: IF FAIL THEN 1090 ELSE B[1,P]=VAL(Z$)
1140 CLS #2: NEXT
1150 R=0: CLS
1160 FOR P=R TO INT((N-1)/10): R=P
1170 FOR Q=1 TO 10: QP=Q+10*P: Q$=FN A$(QP)
1180 LOCATE 4,Q+3: PRINT BS$;"x";Q$;"=";B[0,QP]
1190 LOCATE 20,Q+3: PRINT BS$;"y";Q$;"=";B[1,QP]
1200 NEXT
1210 LOCATE #1,2,1: INPUT #1,"Are these data correct (y/n)";
K$: CLS #1
```

```
1220 K$=UPPER$(K$): IF K$="Y" THEN 1360
1230 LOCATE #1,2,1: PRINT #1,"Which line between";1+10*p;"an
d";10*(P+1);" would you like to alter";: INPUT #2,L: CLS #1:
 CLS #2
1240 IF L>=1+10*P AND L<=10*(P+1) THEN 1260 ELSE LOCATE #1,2
,1: PRINT #1,"Wrong line, try again"
1250 INPUT #2,L: CLS #1: CLS #2: GOTO 1240
1260 LOCATE #1,2,1: PRINT #1,"Enter the correct values"
1270 LOCATE #2,5,1: PRINT #2,"x";FN A$(L);"=";: Z$=""
1280 T$=INKEY$: IF T$="" THEN 1280
1290 PRINT #2,T$;: IF T$<>CHR$(13) THEN Z$=Z$+T$: GOTO 1280
1300 GOSUB 1400: IF FAIL THEN CLS #2: GOTO 1270 ELSE B[0,L]=
VAL(Z$)
1310 LOCATE #2,5,3: PRINT #2,"y";FN A$(L);"=";: Z$=""
1320 T$=INKEY$: IF T$="" THEN 1320
1330 PRINT #2,T$;: IF T$<>CHR$(13) THEN Z$=Z$+T$: GOTO 1320
1340 GOSUB 1400: IF FAIL THEN CLS #2: GOTO 1310 ELSE B[1,L]=
VAL(Z$)
1350 CLS #1: CLS #2: GOTO 1160
1360 NEXT
1370 FOR P=1 TO N
1380 E[0,P]=B[0,P]: E[1,P]=B[1,P]: NEXT
1390 ERASE B: RETURN
1400 FAIL=0: S=0: T=1: IF LEFT$(Z$,1)="+" OR LEFT$(Z$,1)="-"
 THEN T=2
1410 IF Z$="+" OR Z$="-" OR Z$="." THEN 1460
1420 FOR A=T TO LEN(Z$): X$=MID$(Z$,A,1)
1430 IF X$="." THEN S=S+1
1440 IF X$<>" " AND X$<>"." AND (X$<"0" OR X$>"9") THEN 1460
1450 NEXT: IF S<=1 THEN RETURN
1460 FAIL=-1: RETURN
```

21. DISPLAY FILE

The relationship between the memory map and the display on the screen is quite complicated and makes direct PEEKing and POKEing into RAM something to be undertaken with great care. The following simple program shows a lot about the way the display works, and the complications.

```
10 MODE 1
20 Z=&C000
30 FOR P=0 TO 255
40 POKE Z,P: Z=Z+1
50 NEXT: IF Z<-128 THEN 30
60 END
```

Firstly, the initial screen origin Z is the top left of the screen. Successive bytes fill the top line of each printing position over the screen. There are 80 bytes across the screen in each mode and with 25 lines this uses 25x80=2000 bytes. The memory is divided into 8 blocks each 2048 bytes long so that the last 48 bytes of each block are not used. A new block fills the second line of the printing position and so on. The diagram makes this clearer.

In MODE 1, four colours are present but in MODE 2 these are masked down to two. In MODE 0 (change line 10) 16 colours are present with some flashing. The way the colours are controlled is by reference to which bits are 0 and which are 1.

The diagram shows which bits are involved with which pixels in the three modes. In MODE 2, one pixel is associated with one bit so that only two colours are possible as a bit can only be off (ink 0) or on (ink 1) giving background and foreground colours. In MODE 1, the pixels are twice as big and are controlled by two bits. Thus four colours are possible. The following table shows the relationships.

INK 0 both bits off
INK 1 high bit on, low bit off
INK 2 high bit off, low bit on
INK 3 both bits on

These correspond to the binary representation of 0–3. A similar arrangement applies to MODE 0 where 16 colours are possible corresponding to the binary form of 0–15.

By exciting the three phosphors red, blue and green at different intensities a palette of 27 colours is produced.

The second difficulty with the screen display is that the screen scrolls by moving the origin of the screen relative to &C000, the Early Morning Start–up origin. The following program shows a direct POKE to produce an underline.

```
10 CLS
20 LOCATE 10,10
30 PRINT "QWERTY"
40 FOR N=0 TO 11
50 POKE N+&C000+818,240
60 NEXT: PRINT
70 STOP
```

However, scrolling and then re–running separates the underline from the word and it is necessary to re–establish the screen origin at &C000 with machine code routines SCR SET BASE and SCR SET OFFSET. This can be done with the following lines.

```
1000 RESTORE 1000: DATA &E5,&F5,&E3,&C0,
&CD,&08,&BC,&21,&00,&00,&CD,&05,
&BC,&E1,&F1,&C9
1010 MEMORY 40000
1020 A=40200
1030 FOR N=0 TO 15
1040 READ Z
1050 POKE A+N,Z
1060 NEXT
```

Run the machine code program first (with `RUN 1000`) to set up the subroutine then `RUN 10`. After scrolling, `RUN 10` again. Type in `CALL 40200` to restore the original conditions. These techniques may be useful for special effects but not for normal use.

The other aspect of RAM memory is that the user graphics are located just above HIMEM so that it is possible to POKE directly into RAM rather than using SYMBOL. This is illustrated in 'DOUBLE SIZE PRINTING'.

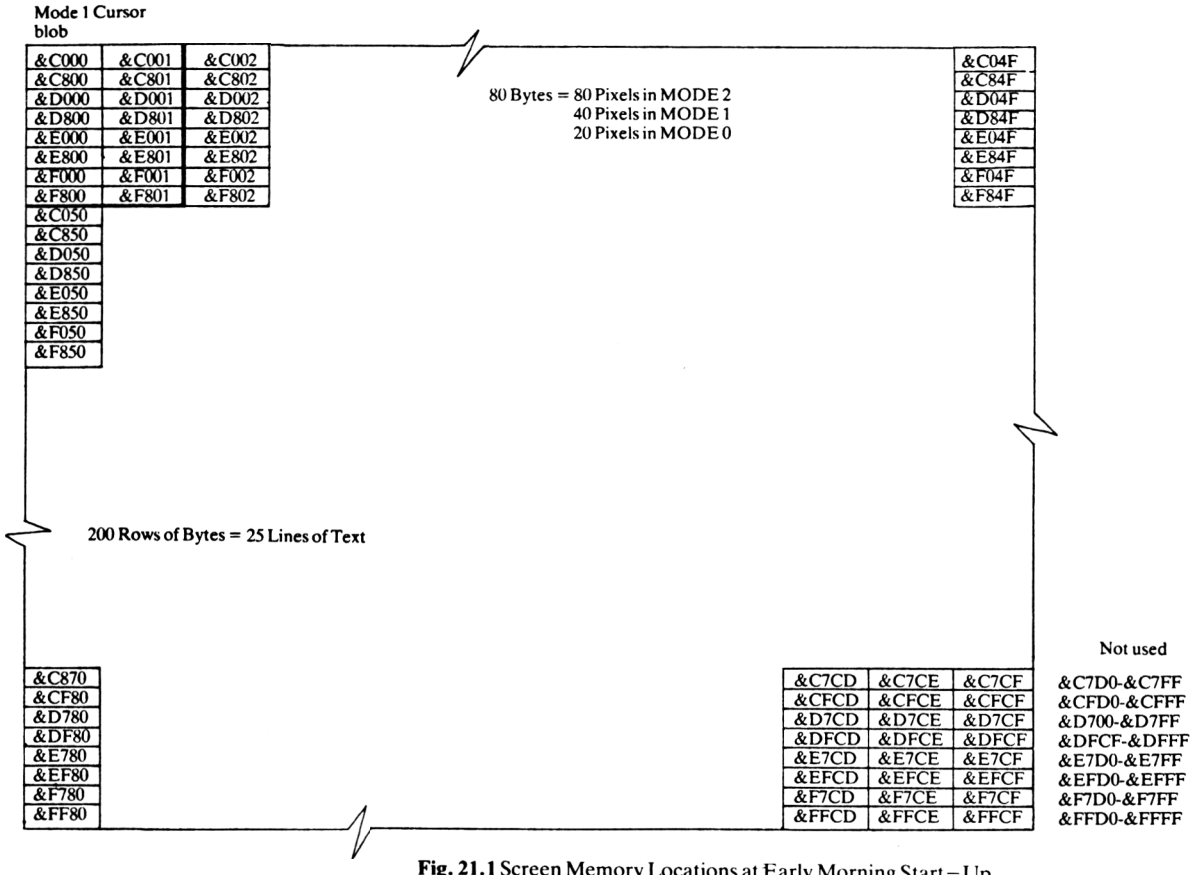


Fig. 21.1 Screen Memory Locations at Early Morning Start – Up

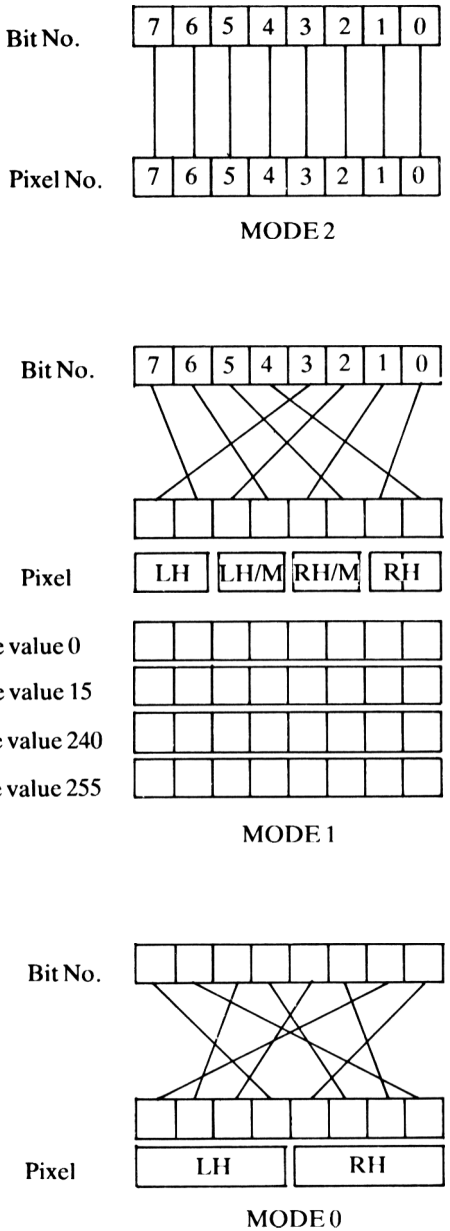


Fig. 21.2 Relationships Between Bits and Colours in each Mode

22. DOUBLE SIZE PRINTING

The Symbol after 64 puts the ASCII information on the capital letters immediately after HIMEM. The program works by PEEKing into successive groups of eight bytes representing the capital letters, at location ADDR. The mathematics converts this information into 32 byte patterns and they are stored in groups of four in the second half of the ASCII set which contains the geometric patterns and foreign letters.

When a capital letter is required in double size, the program first checks that it is a letter and that there is room to print it. If so, it PRINTs four bytes representing the quarters of the letter in two adjacent positions on one line and the other two on the line immediately below. Spaces are separated out and PRINTed with CHR$(32).

It takes about 40 seconds to set up the initial user-defined graphics information.

```
10 REM Double size printing (CAPITAL LETTERS and spaces only)
20 GOSUB 1000: GOSUB 2000: END
1000 SYMBOL AFTER 64: H=HIMEM+1: H1=H+512
1010 DIM T[1,16]
1020 FOR P=1 TO 16: T[0,P]=P-1: READ T[1,P]: NEXT
1030 RESTORE 1030: DATA 0,3,12,15,48,51,60,63,192,195,204,20
7,240,243,252,255
1040 FOR A=1 TO 26: ADDR=H+8*A: B=A*32+H1
1050 FOR P=0 TO 7: T=PEEK(ADDR+P): I=INT(T/16): J=T-16*I
1060 FOR Z=1 TO 16
1070 IF T[0,Z]=J THEN T1=T[1,Z]
1080 IF T[0,Z]=I THEN T2=T[1,Z]
1090 NEXT
1100 C=B+2*P
1110 POKE C,T2: POKE C+1,T2: POKE C+16,T1: POKE C+17,T1
1120 NEXT P,A
1130 RETURN
2000 CLS: PRINT "Enter the matter to be printed": PRINT: INP
UT A$
2010 A$=UPPER$(A$)
2020 FOR P=1 TO LEN(A$): Z=ASC(MID$(A$,P,1))
2030 IF (Z<65 OR Z>90) AND Z<>32 THEN PRINT "  Not capital l
etters. Press any key to re-enter the print material": GOSUB
 2170: GOTO 2000
2040 NEXT
2050 PRINT "Enter the locations where the matter is to be pr
inted"
2060 PRINT: INPUT "Row Number (1-23)   ";R: IF R<1 OR R>23 T
HEN 2060
2070 PRINT: INPUT "Column Number (1-39)";C: IF C<1 OR C>39 T
HEN 2070
```

```
2080 R=2*INT(R/2)+1: C=2*INT(C/2)+1
2090 IF LEN(A$)>240-(C-1)/2-10*(R-1) THEN PRINT " Matter too
 long to print in the screen area. Press any key to re-enter
": GOSUB 2170: GOTO 2000
2100 X=C: Y=R: CLS
2110 FOR P=1 TO LEN(A$): Z=(ASC(MID$(A$,P,1))-64)*4
2120 LOCATE X,Y: IF Z<>-128 THEN PRINT CHR$(Z+128);CHR$(Z+13
0) ELSE PRINT CHR$(32)
2130 LOCATE X,Y+1: IF Z<>-128 THEN PRINT CHR$(Z+129);CHR$(Z+
131) ELSE PRINT CHR$(32)
2140 X=X+2: IF X>39 THEN X=1: Y=Y+2
2150 NEXT
2160 RETURN
2170 IF INKEY$="" THEN 2170
2180 RETURN
```

23. DRAWING LINES BETWEEN TWO POINTS

Drawing lines is very simple with Amstrad BASIC using PLOT and DRAW (or PLOTR and DRAWR) commands. Provided that the coordinates lie within the 16 bit capacity (65535) then the system automatically draws the line which will appear on the screen if the screen area (0<=x<=639, 0<=y<=399) contains all or part of the line.

However, it is important to remember that the aspect ratio of the screen is 0.92 so that if you wish to draw any figure as its true shape, you must allow for this in the coordinates. For example,

```
PLOT 150,40
DRAWR 350,0: DRAWR 0,322
DRAWR -350,0: DRAWR 0,-322
```

draws a true square on the screen even though the number of pixels is different. They are in effect slightly rectangular in shape. In drawing an ellipse, the equation has to take account of the aspect ratio of the circle which gives the ellipse as well as that of the screen.

To draw a dotted line is a little more difficult as two problems arise. Firstly, dealing with either horizontal or vertical lines with a single routine will produce "Division by zero" with one of them. So, two separate routines are needed. The second problem is to keep the mark/space ratio the same for lines at different angles.

Line 2070 caters for the horizontal line and line 2030 for the vertical one. P is the slope but if P is less than one then the reciprocal is used. I is the interval between each new beginning of the mark and a loop is set up in Q to PLOT X1, Y1 then to DRAW 0.4 of the distance to the next PLOT point and so on. A mark/space ratio of 0.4/0.6 looks better than 0.5/0.5.

The SGN in I takes care of the direction of plotting i.e. L to R or R to L and SQR(1+P*P) keeps the mark/space ratio constant for different line slopes.

This subroutine draws a dotted line between (X1,Y1) and (X2,Y2).

```
10 REM Dotted line
20 INPUT "X1=";X1: INPUT "Y1=";Y1: INPUT "X2=";X2: INPUT "Y2
="";Y2
30 CLS: GOSUB 2000: END
2000 IF X1=X2 THEN 2070 ELSE IF Y1=Y2 THEN 2030
```

```
2010 P=(Y2-Y1)/(X2-X1)
2020 IF ABS(P)<1 THEN 2040 ELSE P=1/P: GOTO 2080
2030 P=0
2040 I=10*SGN(X2-X1)/SQR(1+P*P)
2050 FOR Q=0 TO (X2-X1)/I
2060 PLOT X1+Q*I,Y1+Q*I*P: DRAWR 0.4*I,0.4*I*P: NEXT: RETURN
2070 IF Y1=Y2 THEN RETURN ELSE P=0
2080 I=10*SGN(Y2-Y1)/SQR(1+P*P)
2090 FOR Q=0 TO (Y2-Y1)/I
2100 PLOT X1+Q*I*P,Y1+Q*I: DRAWR 0.4*I*P,0.4*I: NEXT: RETURN
```

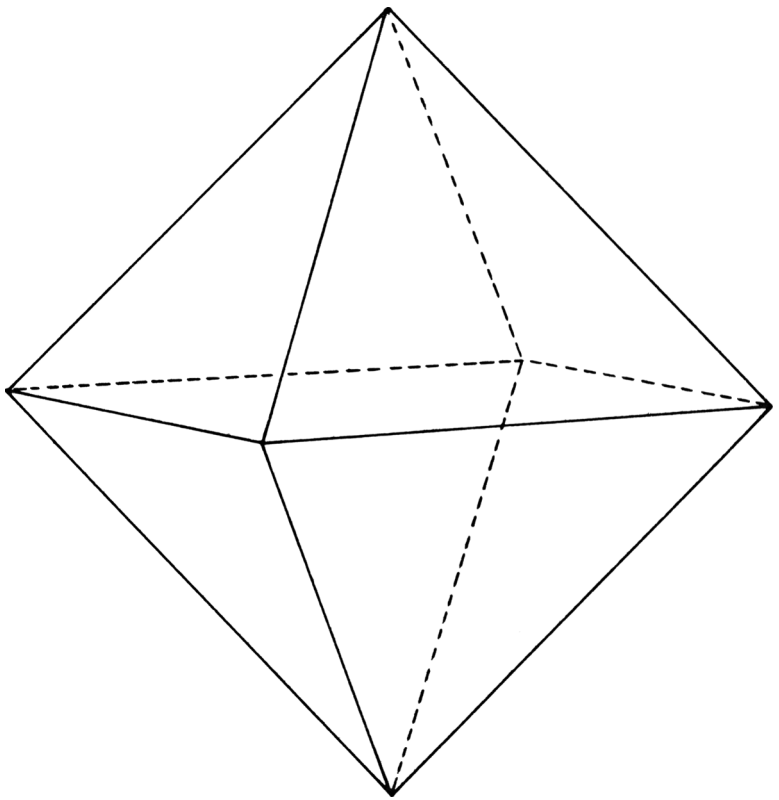


Fig. 23.1 Illustration of Dotted Line Draw

24–26. ERRORS

If you wish to simulate data which follow a known error law then it is desirable to be able to generate numbers which relate to particular distributions. For example, for circumstances where lots of small random errors combine together say, when making a measurement of length, the Gaussion distribution is likely to be the one nearest to the observed results. If the probability of an event happening is very small, such as the number of fatal road accidents occurring in a given period (compared with the total number of journeys made in that period) then the Poisson distribution is much more relevant. Selections of dice throws or hands of cards follow a Binomial distribution.

The random number generator facility in the computer is a rectangular distribution so that any number in the range selected is equally likely to occur. These subroutines alter this distribution so that numbers are more likely to be near the mean of the appropriate distribution rather than at the tails.

The graph shows the typical shapes of the distributions.

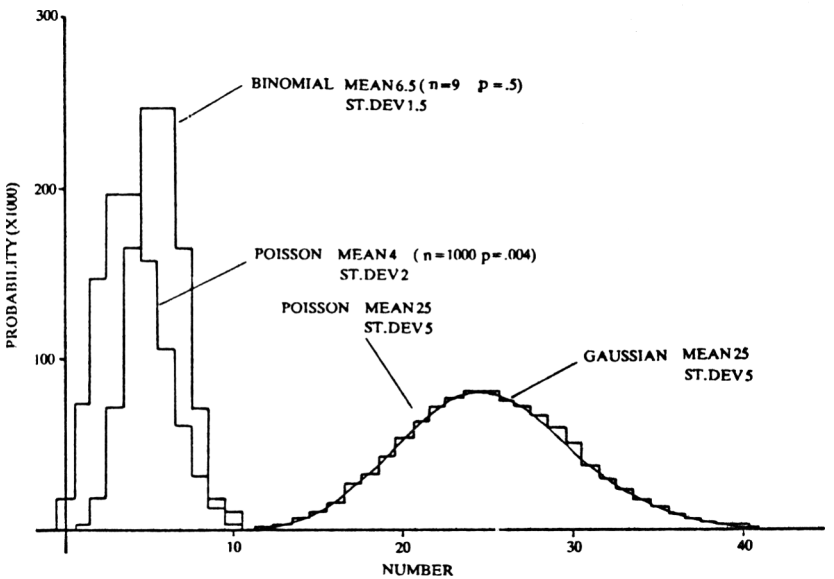


Fig. 24/26.1 Comparison of Different Distributions

24. BINOMIAL DISTRIBUTION

This routine creates an array G which, if sampled randomly, gives a Binomially Distributed set of values. As the size of the array is limited by memory capacity, the routine only works for small values of N as indicated by the expression for Y. The Binomial distribution approximates to the Gaussian distribution as N increases.

```
10 REM Binomial Distribution
20 INPUT "Type in the probability of the event happening, wh
ich must lie in the range 0-1";P
30 IF P<=0 OR P>=1 THEN 20
40 INPUT "Type in the number of items in the selection";N
50 IF N<>ABS(INT(N)) OR N=0 THEN 50
60 GOSUB 1000: GOSUB 2000: END
1000 DIM B[N+3]: T=1/P-1
1010 IF P>0.5 THEN T=1/T
1020 Z=1: M=N
1030 FOR A=1 TO N+1: B[A]=INT(Z+0.5)
1040 Z=Z*M*T/A: M=M-1: NEXT A
1050 Y=INT(1/P^N): IF P>0.5 THEN Y=INT(1/(1-P)^N)
1060 DIM G[Y]: A=1: B=0: Q=B[A]
1070 FOR C=1 TO Y: G[C]=B
1080 IF C>=Q THEN A=A+1: B=B+1: Q=Q+B[A]
1090 NEXT C: IF P<0.5 THEN FOR C=1 TO Y: G[C]=N-G[C]: NEXT C
1100 RETURN
2000 PRINT G[INT(RND(1)*Y)+1];: IF INKEY$="" THEN 2000
2010 RETURN
```

25. GAUSSIAN DISTRIBUTION

This routine sets up an array of 1823 memory locations which, if sampled randomly, will generate values which are Normally Distributed about a chosen mean and have a given standard deviation.

```
10 REM Gaussian Distribution
20 DEF FN G(X)=90.0013*EXP(-X*X/2)
30 DIM G[1823]: GOSUB 1000
40 INPUT "Type in the desired mean value";M
50 INPUT "Type in the desired standard deviation";S
60 IF S<=0 THEN 50
70 GOSUB 2000: END
1000 N=1
1010 FOR P=-3 TO 3 STEP 0.125
1020 E=(FN G(P)+FN G(P+0.125))/2
1030 FOR Q=0 TO E
1040 G[N]=P+RND(1)*0.125: N=N+1
1050 NEXT Q,P
1060 RETURN
2000 Z=M+S*G[INT(RND(1)*1823)+1]
2010 PRINT ROUND(Z,4): IF INKEY$="" THEN 2000
2020 RETURN
```

EXAMPLE

```
run
Type in the desired mean value ? 35
Type in the desired standard deviation ? 1.5
(REM wait)
35.0681
34.0545
34.6708
35.0419
31.6583
33.8876
34.7117
33.6309
33.2394
35.9174
32.6720
36.9178
32.3196
34.8433
36.2537
35.0322
35.5951
*Break*
```

26. POISSON DISTRIBUTION

In the Poisson distribution, the probability of exactly N events happening in a given period is given by

$P=M \uparrow N/N!*EXP(-M)$

where M is the average number of events in the period.

This is the formula used in the subroutine. The standard deviation equals SQR(M) and the distribution is taken out as far as 3*SQR(M) i.e. N3

```
10 REM Poisson Distribution
20 INPUT "Type in the average number of times the event happ
ens in a given period";M: IF M>86 THEN 20
30 GOSUB 1000: CLS: GOSUB 2000: END
1000 M=ABS(M): N3=3*SQR(M)
1010 N=2*N3+(N3-M)*(M<N3)
1020 DIM A[N+1,2]: K=1/M: T=1
1030 FOR P=0 TO M+N3
1040 K=K*M/(P-(P=0))
1050 Z=K*EXP(-M)
1060 IF P>M-N3 THEN A[T,1]=Z: A[T,2]=P: T=T+1
1070 NEXT
1080 DIM B[N+2]: Z=0
1090 FOR P=1 TO N+1
1100 B[P]=INT(A[P,1]*1000+0.5)
1110 Z=Z+B[P]: NEXT
1120 DIM P[Z]: T=1: S=B[T]
1130 FOR P=1 TO Z: P[P]=A[T,2]
1140 IF P>=S THEN T=T+1: S=S+B[T]
1150 NEXT
1160 RETURN
2000 PRINT "Number of times the event happens in subsequent
periods is given by:-"
2010 PRINT P[INT(RND(1)*Z+1)];: IF INKEY$="" THEN 2010
2020 ERASE A,B,P: RETURN
```

27. EVALUATION OF A DETERMINANT

This elegant subroutine is based on the fact that in the Gauss–Jordan method of matrix inversion (see 'MATRIX INVERSION'), the bottom righthand entry in the matrix at the end of the next–to–last cycle of the outer loop is the determinant you want divided by the minor of this entry (i.e. the original determinant with RH column and bottom row omitted). You do not know the value of this minor but it has already been calculated in the same way at the end of the second–to–last cycle of the outer loop, giving the minor of the minor as the unknown. Repeating this process N–1 times leaves you finally with just the top LH entry which you do know. Hence the answer is the product of each of these steps.

The DEF FN is for PRINTing the determinant INPUT entries A[1,1] etc. in packed format.

The first line of the subroutine caters for a determinant with a single term (order 1).

SUB 1140 and SUB 1180 condition the determinant to avoid an unnecessary crash in line 1050 where the 'Division by zero' message would occur if A[I,I]=0 and the protective IF were not present in line 1050.

SUB 1140 ensures that the diagonal entries are not zero and SUB 1180 adds the bottom row to the top row and the RH column to the LH column if the upper LH 2x2 determinant is zero (this does not alter the value of the determinant).

The lines 1030–1120 invert the matrix and multiply the products of each stage up to the N–1 cycle of the I variable.

There is a subtraction in line ten which can produce a zero and this would cause a crash in the next cycle unless detected by line 1110. This aborts the program as the value of the determinant will be zero which happens for example if two rows or two columns are the same or if one is a multiple of the other (see second example).

The subroutine works out the numerical value of a determinant |A| of order N.

```
10 REM Evaluation of a determinant
20 DEFINT I,J,K,N,P,T: DEF FN A$(A)=MID$(STR$(A),2)
30 INPUT "Insert the order of the determinant";N: IF N<1 THE
N 30
40 DIM A[N,N]: FOR I=1 TO N: FOR J=1 TO N
50 PRINT "A["+FN A$(I)+","+FN A$(J)+"]=";: INPUT A[I,J]: NEX
T J,I
60 GOSUB 1000: GOSUB 2000: END
1000 D=A[1,1]: IF N=1 THEN RETURN
1010 GOSUB 1140
1020 IF A[1,1]*A[2,2]=A[1,2]*A[2,1] THEN GOSUB 1180
1030 D=A[1,1]*T: FOR P=N TO 2 STEP -1: DIM D[P,P]
1040 FOR I=1 TO P: FOR J=1 TO P: D[I,J]=A[I,J]: NEXT J,I
1050 FOR I=1 TO P-1: IF ABS(D[I,I])<0.00000001 THEN D=0: RET
URN ELSE D[I,I]=1/D[I,I]
1060 FOR J=1 TO P: IF J=I THEN 1110
1070 D[J,I]=D[J,I]*D[I,I]
1080 FOR K=1 TO P: IF K=I THEN 1100
1090 D[J,K]=D[J,K]-D[J,I]*D[I,K]
1100 NEXT
1110 NEXT: IF ABS(D[I+1,I+1])<0.00000001 THEN D=0: RETURN
1120 NEXT: D=D*D[P,P]: ERASE D: NEXT
1130 RETURN
1140 T=1: FOR I=1 TO N: IF A[I,I]=0 THEN GOSUB 1160
1150 NEXT: RETURN
1160 FOR J=2 TO N: IF A[I,J]<>0 THEN T=-T: FOR K=1 TO N: SW=
A[K,I]: A[K,I]=A[K,J]: A[K,J]=SW: NEXT: RETURN
1170 NEXT: RETURN
1180 FOR I=1 TO N: A[1,I]=A[1,I]+A[N,I]: NEXT
1190 FOR I=1 TO N: A[I,1]=A[I,1]+A[I,N]: NEXT
1200 RETURN
2000 PRINT "Det D=";D
2010 RETURN
```

EXAMPLES

N=4

$$\begin{vmatrix} -2 & 4 & 7 & 3 \\ 8 & 2 & -9 & 5 \\ -4 & 6 & 8 & 4 \\ 2 & -9 & 3 & 8 \end{vmatrix}$$

Det D=2140

N=5

$$\begin{vmatrix} 15 & -7 & 6 & 9 & 3 \\ 2 & 4 & 6 & 8 & 10 \\ 31 & 6 & 11 & 17 & 2 \\ 1 & 2 & 3 & 4 & 5 \\ 81 & 9 & 23 & 1 & 3 \end{vmatrix}$$

Det D=0

28. FACTORIAL n

This subroutine calculates factorial n (for n<33). It illustrates a very simple loop with accumulative multiplication. Factorials occur in Permutations and Combinations as well as in series.

```
10 REM Factorial n
20 INPUT "Number";N: IF N<>ABS(INT(N)) OR N>33 THEN 20
30 GOSUB 1000: GOSUB 2000: END
1000 Z=1: FOR P=1 TO N: Z=Z*P: NEXT
1010 RETURN
2000 PRINT "N!=";Z
2010 RETURN
```

29. HEAVISIDE OPERATOR

This subroutine performs the Heaviside operation of turning a function on at x=a and off at x=b.

This is very useful in studying mechanical and electrical problems where step functions occur eg. a switch being turned on. To test the stability of a system or itsresponse, a step function is often used. In the example, the response of a circuit containing a capacitor and a resistance in series is given and it can be seen that even though the voltage is turned on and off instantaneously, the condenser voltage always lags behind. It takes time to charge up and discharge. Change the value of RC to see the effect on the response. This test is often used to assess the performance of amplifiers and loud speakers in Hi−Fi systems.

The Control Stability Program is another example of the use of the Heaviside Operator.

```
10 REM Heaviside operator
20 DEF FN H(X,A,B)=-(X>=A)+(X>=B)
30 REM EXAMPLE
40 A=100: B=198: RC=10
50 DEF FN H(X,A,B)=-(X>=A)+(X>=B)
60 FOR X=1 TO 639: T=(X-100*INT(X/100))/RC
70 PLOT X,150+128*EXP(-T)+128*FN H(X,A,B+2)*(1-2*EXP(-T))
80 PLOT X,128*FN H(X,A+1,B+1)
90 IF X=A-1 OR X=B+1 THEN DRAWR 0,128
100 IF X>B THEN A=A+200: B=B+200
110 NEXT
```

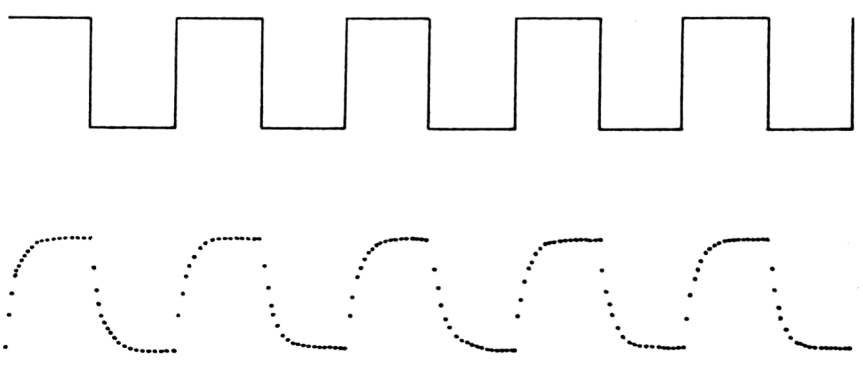


Fig. 29.1 Square Wave and Damped Square Wave

30–33. HIGH PRECISION ARITHMETIC

As simple calculations with the Amstrad are accurate to one part in ten to the ninth, there seems little need for anything more accurate. However, this group of programs is included as they are excellent examples of string handling techniques and cover all the possibilities. There are several ways of approaching calculations using strings, the ones adopted here for add, subtract and multiply follow the longhand calculation methods. They also use look–up tables to find the tens and units e.g. "9" + "7" = "1" ten and "6" units.

Reciprocal is different and is based on a combination of the other three using a successive approximation method. It is very slow in BASIC but in machine code is an excellent way of doing integer division.

The following are the look–up tables generated in the first sections of the routines.

TENS AND UNITS LOOK–UP TABLES

P + Q

P \ Q	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	0
2	2	3	4	5	6	7	8	9	0	1
3	3	4	5	6	7	8	9	0	1	2
4	4	5	6	7	8	9	0	1	2	3
5	5	6	7	8	9	0	1	2	3	4
6	6	7	8	9	0	1	2	3	4	5
7	7	8	9	0	1	2	3	4	5	6
8	8	9	0	1	2	3	4	5	6	7
9	9	0	1	2	3	4	5	6	7	8

UNITS

P \ Q	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	0	1	1
3	0	0	0	0	0	0	0	1	1	1
4	0	0	0	0	0	0	1	1	1	1
5	0	0	0	0	0	1	1	1	1	1
6	0	0	0	0	1	1	1	1	1	1
7	0	0	0	1	1	1	1	1	1	1
8	0	0	1	1	1	1	1	1	1	1
9	0	1	1	1	1	1	1	1	1	1

TENS

P – Q

P \ Q	0	1	2	3	4	5	6	7	8	9
0	0	9	8	7	6	5	4	3	2	1
1	1	0	9	8	7	6	5	4	3	2
2	2	1	0	9	8	7	6	5	4	3
3	3	2	1	0	9	8	7	6	5	4
4	4	3	2	1	0	9	8	7	6	5
5	5	4	3	2	1	0	9	8	7	6
6	6	5	4	3	2	1	0	9	8	7
7	7	6	5	4	3	2	1	0	9	8
8	8	7	6	5	4	3	2	1	0	9
9	9	8	7	6	5	4	3	2	1	0

UNITS

P \ Q	0	1	2	3	4	5	6	7	8	9
0	0	1	1	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	1	1	1
2	0	0	0	1	1	1	1	1	1	1
3	0	0	0	0	1	1	1	1	1	1
4	0	0	0	0	0	1	1	1	1	1
5	0	0	0	0	0	0	1	1	1	1
6	0	0	0	0	0	0	0	1	1	1
7	0	0	0	0	0	0	0	0	1	1
8	0	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	0	0	0

TENS

P * Q

P \ Q	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	0	2	4	6	8
3	0	3	6	9	2	5	8	1	4	7
4	0	4	8	2	6	0	4	8	2	6
5	0	5	0	5	0	5	0	5	0	5
6	0	6	2	8	4	0	6	2	8	4
7	0	7	4	1	8	5	2	9	6	3
8	0	8	6	4	2	0	8	6	4	2
9	0	9	8	7	6	5	4	3	2	1

UNITS

P \ Q	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	1	1	1	1	1
3	0	0	0	0	1	1	1	2	2	2
4	0	0	0	1	1	2	2	2	3	3
5	0	0	1	1	2	2	3	3	4	4
6	0	0	1	1	2	3	3	4	4	5
7	0	0	1	2	2	3	4	4	5	6
8	0	0	1	2	3	4	4	5	6	7
9	0	0	1	2	3	4	5	6	7	8

TENS

30. HIGH PRECISION ARITHMETIC – Addition

FN L(A,B) finds which is the larger A or B and the two loops from 0 to 9 work out the contents of the look–up table.

After the INPUT, the position of the decimal point is found and the number turned inside out (see Subtraction).

D$[1] is filled with zeros using the STRING$ function because this defines the minimum length of string necessary to hold the number (equal to the larger integer part + the larger decimal part +1).

In the next four lines, MID$ is used with its statement meaning to replace some of the zeros in D$ with X$ and Y$. The addition is done in SUB 1200 and 1240 using the VAL of P$ and Q$ to find the correct answers. Note that P$ and Q$ are defined using MID$ as a function to return a specified part of D$.

C is the carry and in the PRINT, (S[RL−L+1]=0) is used to suppress the leading zero, if present.

```
10 REM High precision arithmetic-Addition (Positive nmbers o
nly)
20 DEFINT A,B,C,L,P,Q,R,X,Y,Z
30 DEF FN L(A,B)=A-(B-A)*(B>A)
40 DEF FN A$(A)=MID$(STR$(A),2)
50 DIM A$[9,9],B$[9,9]
60 FOR P=0 TO 9: FOR Q=0 TO 9
70 A$[P,Q]=MID$(STR$(100+P+Q),4,1)
80 B$[P,Q]=MID$(STR$(100+P+Q),3,1)
90 NEXT Q,P
100 LINE INPUT "X=";X$: LINE INPUT "Y=";Y$: IF X$="" OR Y$="
" OR LEFT$(X$,1)="-" OR LEFT$(Y$,1)="-" THEN 100
110 GOSUB 1000: GOSUB 2000
120 ERASE D$,S: END
1000 LX=LEN(X$): LY=LEN(Y$)
1010 XL=INSTR(X$,".")-1: YL=INSTR(Y$,".")-1
1020 IF XL=-1 THEN XL=LX
1030 IF YL=-1 THEN YL=LY
1040 XR=LX-XL+(LX<>XL): YR=LY-YL+(LY<>YL)
1050 L=FN L(XL,YL): R=FN L(XR,YR): RL=R+L
1060 DIM D$[1]: FOR P=0 TO 1
1070 D$[P]=STRING$(RL,48): NEXT
1080 IF XR<>0 THEN MID$(D$[0],1)=RIGHT$(X$,XR)
1090 IF YR<>0 THEN MID$(D$[1],1)=RIGHT$(Y$,YR)
1100 IF XL<>0 THEN MID$(D$[0],RL-XL+1)=LEFT$(X$,XL)
1110 IF YL<>0 THEN MID$(D$[1],RL-YL+1)=LEFT$(Y$,YL)
1120 DIM S[RL+1]: C=0
1130 IF R<>0 THEN FOR P=R TO 1 STEP -1: P$=MID$(D$[0],P,1):
Q$=MID$(D$[1],P,1): GOSUB 1200: NEXT
1150 IF L<>0 THEN FOR P=RL TO RL-L+1 STEP -1: P$=MID$(D$[0],
P,1): Q$=MID$(D$[1],P,1): GOSUB 1240: NEXT: S[RL-L+1]=C
1170 RETURN
1200 S[P]=VAL(A$[VAL(P$),C]): C=VAL(B$[VAL(P$),C]): Z=S[P]
1210 S[P]=VAL(A$[Z,VAL(Q$)]): C=C+VAL(B$[Z,VAL(Q$)])
1220 RETURN
1240 S[P+1]=VAL(A$[VAL(P$),C]): C=VAL(B$[VAL(P$),C]): Z=S[P+
1]
```

```
1250 S[P+1]=VAL(A$[Z,VAL(Q$)]): C=C+VAL(B$[Z,VAL(Q$)])
1260 RETURN
2000 IF L<>0 THEN FOR P=RL-L+1-(S[RL-L+1]=0) TO RL+1: PRINT
FN A$(S[P]);: NEXT
2010 IF L=0 THEN PRINT FN A$(C);
2020 IF R<>0 THEN PRINT ".";: FOR P=1 TO RL-L: PRINT FN A$(S
[P]);: NEXT
2030 RETURN
```

EXAMPLE

```
run
X=123456789.987654321
Y=12121212112.1212121
12244668902.108866421
```

31. HIGH PRECISION ARITHMETIC – Subtraction

The first group of lines fills in the subtraction look–up table. The –20*(P<Q) is to convert the 9 from borrowing to a 1 for a carry as in the addition routine (see tables).

The FN L(A,B) defines which is larger, A or B and the FN A$(A) is for packed format in the string printout.

After the INPUT with SIGN$ taking care of the sign, the position of the decimal point is found with INSTR(X$,".") and for convenience the numbers are turned inside out (lines 1080–1110).

For example, 123.45678
–45.678

becomes 45678123
–67800045

having first filled D$ with zeros.

The subtraction is done in two separate loops using the SUB 1160 and the PRINT routines reconstitute the answer with its decimal point and correct sign. Note the use of Z in the subroutine to preserve the correct value of D[P] in the last statement and not the value of D[P] from the next–to–last statement.

```
10 REM High precision arithmetic-Subtraction (Positive nmber
s only)
20 DEFINT A,B,C,L,P,Q,R,X,Y,Z
30 DEF FN L(A,B)=A-(B-A)*(B>A)
40 DEF FN A$(A)=MID$(STR$(A),2)
50 DIM S$[9,9],T$[9,9]
60 FOR P=0 TO 9: FOR Q=0 TO 9
70 S$[P,Q]=MID$(STR$(100+P-Q-20*(P<Q)),4,1)
80 T$[P,Q]=MID$(STR$(100+P-Q-20*(P<Q)),3,1)
90 NEXT Q,P
100 SIGN$="+": LINE INPUT "X=";X$: LINE INPUT "Y=";Y$: IF X$
=="" OR Y$="" OR LEFT$(X$,1)="-" OR LEFT$(Y$,1)="-" THEN 100
110 IF VAL(X$)<VAL(Y$) THEN SW$=X$: X$=Y$: Y$=SW$: SIGN$="-"
120 GOSUB 1000: GOSUB 2000
130 ERASE D$,D: END
1000 LX=LEN(X$): LY=LEN(Y$)
1010 XL=INSTR(X$,".")-1: YL=INSTR(Y$,".")-1
1020 IF XL=-1 THEN XL=LX
1030 IF YL=-1 THEN YL=LY
1040 XR=LX-XL+(LX<>XL): YR=LY-YL+(LY<>YL)
1050 L=FN L(XL,YL): R=FN L(XR,YR): RL=R+L
1060 DIM D$[1]: FOR P=0 TO 1
1070 D$[P]=STRING$(RL,48): NEXT
```

```
1080 IF XR<>0 THEN MID$(D$[0],1)=RIGHT$(X$,XR)
1090 IF YR<>0 THEN MID$(D$[1],1)=RIGHT$(Y$,YR)
1100 IF XL<>0 THEN MID$(D$[0],RL-XL+1)=LEFT$(X$,XL)
1110 IF YL<>0 THEN MID$(D$[1],RL-YL+1)=LEFT$(Y$,YL)
1120 DIM D[RL]: C=0
1130 IF R<>0 THEN FOR P=R TO 1 STEP -1: GOSUB 1160: NEXT
1140 IF L<>0 THEN FOR P=RL TO RL-L+1 STEP -1: GOSUB 1160: NE
XT
1150 RETURN
1160 P$=MID$(D$[0],P,1): Q$=MID$(D$[1],P,1)
1170 D[P]=VAL(S$[VAL(P$),C]): C=VAL(T$[VAL(P$),C]): Z=D[P]
1180 D[P]=VAL(S$[Z,VAL(Q$)]): C=C+VAL(T$[Z,VAL(Q$)])
1190 RETURN
2000 PRINT SIGN$;:IF L<>0 THEN FOR P=1+R TO RL: PRINT FN A$(
D[P]);: NEXT
2010 IF R<>0 THEN PRINT ".";: FOR P=1 TO R: PRINT FN A$(D[P]
);: NEXT
2020 RETURN
```

EXAMPLE

```
X = 135791357913579
Y = 24682468.2468246

+135791333231110.7531754
```

32. HIGH PRECISION ARITHMETIC – Multiplication

The first part follows the addition and subtraction routines closely but the decimal point is removed from the number if present, rather than turning the number inside out.

After the DIM statements, the multiplication is performed with the look–up table but the tens and units are kept separate and the sum of each column is stored in M[L,1] and M[L,2]. As D$ contains some blank spaces from the DIM operation, these have to be avoided before the proper entries can be added together in P[L] using the VAL function.

The 0.00001 is added to avoid rounding errors.

```
10 REM High precision arithmetic-Multiplication
20 DEFINT A,B,C,L,M,P,Q,X,Y
30 DEF FN A$(A)=MID$(STR$(A),2)
40 DIM M$[9,9],N$[9,9]
50 FOR P=0 TO 9: FOR Q=0 TO 9
60 M$[P,Q]=MID$(STR$(100+P*Q),4,1)
70 N$[P,Q]=MID$(STR$(100+P*Q),3,1)
80 NEXT Q,P
90 LINE INPUT "X=";X$: LINE INPUT "Y=";Y$:
100 IF X$="" OR Y$="" OR LEFT$(X$,1)="-" OR LEFT$(Y$,1)="-"
THEN 90
110 GOSUB 1000: GOSUB 2000: END
1000 XD=INSTR(X$,".")-1: YD=INSTR(Y$,".")-1
1010 IF XD=-1 THEN XD=LEN(X$): GOTO 1030
1020 X$=LEFT$(X$,XD)+MID$(X$,XD+2)
1030 IF YD=-1 THEN YD=LEN(Y$): GOTO 1050
1040 Y$=LEFT$(Y$,YD)+MID$(Y$,YD+2)
1050 LX=LEN(X$): LY=LEN(Y$): L=LX+LY: LD=L-XD-YD
1060 DIM D$[L,LY,1],M[L,2],P[L]
1070 FOR P=1 TO LX: FOR Q=1 TO LY
1080 A=VAL(MID$(X$,P,1)): B=VAL(MID$(Y$,Q,1))
1090 D$[P+Q,Q,0]=M$[A,B]: D$[P+Q-1,Q,1]=N$[A,B]
1100 NEXT Q,P
1110 FOR P=L TO 1 STEP -1: FOR Q=1 TO LY
1120 IF D$[P,Q,0]<>" " THEN M[P,1]=M[P,1]+VAL(D$[P,Q,0])
1130 IF D$[P,Q,1]<>" " THEN M[P,2]=M[P,2]+VAL(D$[P,Q,1])
1140 NEXT Q,P
1150 C=0: FOR P=L TO 1 STEP -1
1160 M=C+M[P,1]+M[P,2]: C=0
1170 IF M>9 THEN C=INT(M/10+0.00001): M=INT(10*(M/10-INT(M/1
0))+0.00001)
1180 P[P]=M: NEXT
1190 RETURN
2000 IF L<>LD THEN FOR P=1-(P[1]=0) TO L-LD: PRINT FN A$(P[P
]);: NEXT
```

```
2010 IF LD<>0 THEN PRINT ".";: FOR P=L-LD+1 TO L: PRINT FN A
$(P[P]);: NEXT
2020 ERASE D$,M,P: RETURN
```

EXAMPLES

```
run
X=1234.56789
Y=98765.4321
121932631.112635269

run
X=999999999999
Y=999999999999
99999999999800000000001
```

33. HIGH PRECISION ARITHMETIC – Reciprocal

This program is the most interesting of the group. It is based on the fact that if A is the reciprocal of P, where P is an integer, then obviously
$A=1/P$

If A_i is the ith approximation to 1/P then the exact value of A is given by

$$A=A_i+(1/P)*(1-A_i P)$$

If we substitute A_i for 1/P then

$$A_{i+1}=A_i+A_i(1-A_iP)$$

$$=2A_i-A_i^2P$$

As we are dealing with integers, it is necessary to remove the decimal point, if present, by multiplying by a suitable power of 10. The program then finds 200 times A (200 corresponds to 2A with two extra digits) and subtracts $A^2 P$ which gives the next approximation two digits better.

The first block works out the six look–up tables required.

After the number is INPUT as I$, it is checked for "+" and "–", and the decimal place is sought in 200 or 220 depending on whether the number is greater than or less than one. E$ is the first approximation to the answer which is an integer representing the first 8 non–zero digits in the reciprocal calculated by the computer.

E$ is sent to SUB Times 200 and then to Multiply to produce the square. This latter is sent back to Multiply with C$ to produce A^2 P and then the subtraction is performed in SUB Subtract to produce the next value of E$.

The routine can be speeded up by only considering what happens beyond the places in the strings where the digits change each time. If you want to become proficient in string handling, work out the modifications needed!.

It is interesting to attempt to SAVE this program. Nothing happens for 40 seconds whilst the computer reorganises its memory and gets rid of the garbage. This happens also when you ask for FRE("") which gives the amount of free string space.

EXAMPLE.

```
run
X=17
Precison Required? 20
5.88235294117647058823 6E-2
Ready
```

Inserting a PRINT E$ just inside the I loop shows how the number builds up.

```
run
X=17
Precision Required? 20
58823529
5882352942
588235294118
58823529411765
5882352941176471
58823529411764705 9
588235294117647058 3
5.88235294117647058823 6E-2
```

Note that the last digit is always one digit high. Also, like many prime numbers, but not all, it gives a recurring decimal which recurs after a number of decimal places equal to the prime number itself, in this case 17.

```
10 REM High precision arithmetic-Reciprocal
20 DEFINT C,I,K,L,N,P,Q,T,V,Z
30 DEF FN A$(A)=MID$(STR$(A),2)
40 DIM A$[9,9],B$[9,9],M$[9,9],N$[9,9],S$[9,9],T$[9,9]
50 FOR P=0 TO 9: FOR Q=0 TO 9
60 A$[P,Q]=MID$(STR$(100+P+Q),4,1)
70 B$[P,Q]=MID$(STR$(100+P+Q),3,1)
80 M$[P,Q]=MID$(STR$(100+P*Q),4,1)
90 N$[P,Q]=MID$(STR$(100+P*Q),3,1)
100 S$[P,Q]=MID$(STR$(100+P-Q-20*(P<Q)),4,1)
110 T$[P,Q]=MID$(STR$(100+P-Q-20*(P<Q)),3,1)
120 NEXT Q,P
130 T=1: LINE INPUT "X=";I$: IF I$="" THEN 130
140 IF VAL(I$)=0 THEN 130
150 IF LEFT$(I$,1)="+" OR LEFT$(I$,1)="-" THEN T=2
160 ID=INSTR(I$,".")-1: IF ID=-1 THEN ID=LEN(I$): N=0: GOTO
180
170 N=LEN(I$)-1-ID
180 X=ABS(VAL(I$)): IF X=1 THEN GOSUB 2020: END
190 M=X: K=0: IF X<1 THEN K=1: GOTO 220
200 WHILE M>1: M=M/10: K=K+1: WEND
210 GOTO 230
220 WHILE M<1: M=M*10: K=K-1: WEND
```

```
230 E$=MID$(STR$(INT(10^(K+7)/X)),2)
240 C$=MID$(I$,T,ID)+MID$(I$,ID+2)
250 INPUT "Precision Required"; C: C=ABS(INT(C)): IF C<9 THE
N 250
260 GOSUB 1000: GOSUB 2000: END
1000 FOR I=1 TO C-7 STEP 2
1010 X$=E$: GOSUB 1500: REM Times 200
1020 Y$=X$: GOSUB 1600: REM Multiply
1030 X$=Z$: Y$=C$: GOSUB 1600: REM Multiply
1040 X$=V$: Y$=LEFT$(Z$,LEN(Z$)-4-K-N-I): GOSUB 1800: REM Su
btract
1050 NEXT
1060 RETURN
1499 REM Times 200
1500 LX=LEN(X$): DIM S[LX+2]: C=0
1510 FOR P=LX TO 1 STEP -1: VX=VAL(MID$(X$,P,1))
1520 S[P-1]=VAL(A$[VX,C]): C=VAL(B$[VX,C]): Z=S[P-1]
1530 S[P]=VAL(A$[VX,Z]): C=C+VAL(B$[VX,Z])
1540 NEXT: S[0]=C: V$=""
1550 FOR P=0 TO LX+2: V$=V$+FN A$(S[P]): NEXT
1560 ERASE S: RETURN
1599 REM Multiply
1600 LX=LEN(X$): LY=LEN(Y$): L=LX+LY
1610 DIM D$[L,LY,1],M[L,2],P[L]
1620 FOR P=1 TO LX: FOR Q=1 TO LY
1630 VX=VAL(MID$(X$,P,1)): VY=VAL(MID$(Y$,Q,1))
1640 D$[P+Q,Q,0]=M$[VX,VY]: D$[P+Q-1,Q,1]=N$[VX,VY]
1650 NEXT Q,P
1660 FOR P=L TO 1 STEP -1: FOR Q=1 TO LY
1670 IF D$[P,Q,0]<>" " THEN M[P,1]=M[P,1]+VAL(D$[P,Q,0])
1680 IF D$[P,Q,1]<>" " THEN M[P,2]=M[P,2]+VAL(D$[P,Q,1])
1690 NEXT Q,P
1700 C=0: FOR P=L TO 1 STEP -1
1710 M=C+M[P,1]+M[P,2]: C=0
1720 IF M>9 THEN C=INT(M/10+0.00001): M=INT(10*(M/10-C)+0.00
001)
1730 P[P]=M: NEXT: Z$=""
1740 FOR P=1-(P[1]=0) TO L: Z$=Z$+FN A$(P[P]): NEXT
1750 ERASE D$,M,P: RETURN
1799 REM Subtract
1800 LX=LEN(X$): LY=LEN(Y$)
1810 DIM D$[1]: FOR P=0 TO 1
1820 D$[P]=STRING$(LX,48): NEXT
1830 MID$(D$[0],1)=X$: MID$(D$[1],LX-LY+1)=Y$
1840 DIM D[LX]: C=0
1850 FOR P=LX TO 1 STEP -1
1860 VX=VAL(MID$(D$[0],P,1)): VY=VAL(MID$(D$[1],P,1))
1870 D[P]=VAL(S$[VX,C]): C=VAL(T$[VX,C])
1875 Z=D[P]: D[P]=VAL(S$[Z,VY])
1880 C=C+VAL(T$[Z,VY]): NEXT: E$=""
1890 FOR P=1-(D[1]=0) TO LX: E$=E$+FN A$(D[P]): NEXT
1900 ERASE D$,D: RETURN
2000 IF T=2 THEN PRINT LEFT$(I$,1);
2010 PRINT LEFT$(E$,1);".";MID$(E$,2);"E";-K: RETURN
2020 PRINT I$
2030 RETURN
```

34. INORDER SEQUENCE

This subroutine labels the vertices of a binary tree in INORDER sequence. Each vertex to the left of any other vertex nearer the root is labelled with a lower value and each one to the right is higher. The illustration shows a binary tree labelled with the names of computer languages in INORDER sequence lexicographically. (See 'BINARY SEARCH TREE' for related routine.)

```
10 REM INORDER Sequence for words
20 INPUT "Height of tree";K: K=2^(K+2)-1: L=(K+1)/2-1
30 DIM A$[K],A[K],S[K]
40 FOR P=1 TO L: INPUT A$[P]: NEXT
50 GOSUB 1000: GOSUB 2000: END
1000 C=1: V=1: SP=K
1010 WHILE A$[2*V]<>"": GOSUB 1060: V=V+V: WEND
1020 A[V]=C: C=C+1
1030 IF A$[2*V+1]<>"" THEN V=2*V+1: GOTO 1010
1040 IF SP<>K THEN GOSUB 1070: GOTO 1020
1050 RETURN
1060 S[SP]=V: SP=SP-1: RETURN
1070 V=S[SP+1]: SP=SP+1: RETURN
2000 FOR P=1 TO L: PRINT A$[P];TAB(10);A[P]: NEXT
2010 RETURN
```

```
10 REM INORDER Sequence for numbers
20 INPUT "Height of tree";K: K=2^(K+2)-1: L=(K+1)/2-1
30 DIM A[K,1],S[K]
40 FOR P=1 TO L: INPUT A[P,0]: NEXT
50 GOSUB 1000: GOSUB 2000: END
1000 C=1: V=1: SP=K
1010 WHILE A[2*V,0]<>0: GOSUB 1060: V=V+V: WEND
1020 A[V,1]=C: C=C+1
1030 IF A[2*V+1,0]<>0 THEN V=2*V+1: GOTO 1010
1040 IF SP<>K THEN GOSUB 1070: GOTO 1020
1050 RETURN
1060 S[SP]=V: SP=SP-1: RETURN
1070 V=S[SP+1]: SP=SP+1: RETURN
2000 FOR P=1 TO L: PRINT A[P,0];TAB(10);A[P,1]: NEXT
2010 RETURN
```

EXAMPLE

```
run                  Output
Height of tree? 4    MALLARD    11
? MALLARD            BCPL       4
? BCPL               QLSUPER    14
```

```
? QLSUPER              BASIC09    2
? BASIC09              FORTH      7
? FORTH                PASCAL     12
? PASCAL               SPECTRUM   15
? SPECTRUM             ALGOL      1
? ALGOL                BBC        3
? BBC                  COBOL      6
? COBOL                LISP       9
? LISP                            0
?                      PIP        13
? PIP                             0
?                                 0
?                                 0
?                                 0
?                                 0
?                                 0
?                                 0
? C                    C          5
?                                 0
? FORTRAN              FORTRAN    8
? LOGO                 LOGO       10
?                                 0
?                                 0
?                                 0
?                                 0
?                                 0
?                                 0
?                                 0
?                                 0

                       Ready
```

Note that the words are numbered in alphabetical order.

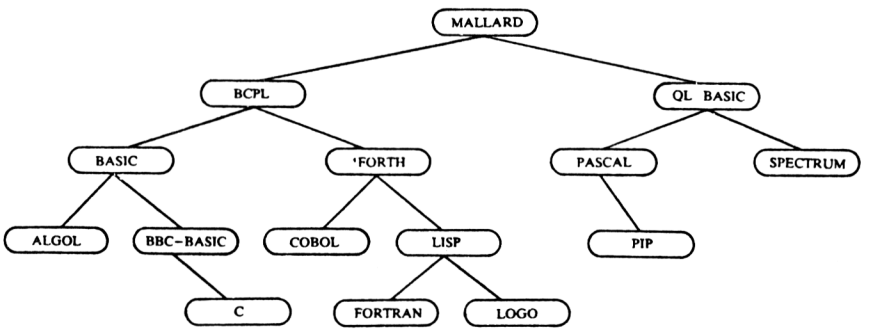


Fig. 34.1 Binary Tree of Computer Languages

35. INTERPOLATION

The idea behind interpolation is to fit a curve of some sort through a set of points so that you can calculate a value at an intermediate position where there is no point. The best way of doing this depends very much on the quality of the data that you are dealing with and there are no hard–and–fast rules to get the best answer.

Obviously, a straight line can always be drawn through two points, a second order curve, (a circle, parabola, ellipse or hyperbola) can be drawn through three points and a cubic curve through four points etc.. However, there is no point in fitting an n^{th} order curve through a set of points if the scatter from a straight line is due to errors in the measurement. It is then better to use a 'Best Fit Line' approach and use the equation of this to predict the value and the likely errors associated with it.

Similarly, if a set of points lies about a curve, it may be more sensible to replot on a log/linear or log/log scale to produce a straight scatter band and then use this for the 'Best Fit Line' approach.

If the data are very accurate then the way to find the polynomial to represent the curve is to select a number of points spaced out along the curve including the first and last in the range of interest and fit a curve of power one less than the number of points i.e. to fit a fourth power polynomial you need five points etc..

This is best illustrated by example. The table shows the Student's t values for N=10 to 60. To fit a cubic equation we require that:–

$aN^3+bN^2+cN+d=t$

choosing the points N=10, 15, 25 and 60 we have:–

```
a*1000   +b*100  +c*10+d=2.228
a*3375   +b*225  +c*15+d=2.131
a*15625  +b*625  +c*25+d=2.060
a*216000+b*3600+c*60+d=2.000
```

and solving these using the 'Simultaneous Equations' subroutine gives the values

```
a=–0.000014
b= 0.001520
c=–0.050755
d= 2.597524
```

From these values the intermediate ones shown in the table have been calculated and the difference column shows how the approximate curve weaves about the true line. Between 10 and 27 the error is less than 0.4% and good enough for the purpose. To get a better fit would require a higher order curve to be fitted.

Student's t Table

N	Actual from Tables	Calculated from equation	Difference (Calc. − Actual)
10	2.228*	(2.228)	−
11	2.201	2.205	+0.004
12	2.179	2.183	+0.004
13	2.160	2.164	+0.004
14	2.145	2.147	+0.002
15	2.131*	(2.131)	−
16	2.120	2.117	−0.003
17	2.110	2.105	−0.005
18	2.101	2.095	−0.006
19	2.093	2.086	−0.007
20	2.086	2.078	−0.008
21	2.080	2.072	−0.008
22	2.074	2.068	−0.006
23	2.069	2.064	−0.005
24	2.064	2.061	−0.003
25	2.060*	(2.060)	−
26	2.056	2.059	+0.003
27	2.052	2.060	+0.008
28	2.048	2.061	+0.013
29	2.045	2.063	+0.018
30	2.042	2.065	+0.026
40	2.021	2.103	+0.082
60	2.000*	(2.000)	−

* Used to establish the coefficients.

Finally a word of warning, don't quote more accuracy in the calculated values than is inherent in the original data and NEVER try to calculate values outside the range you originally used.

36. LABEL

This subroutine is associated with storage of information. The information is in two parts, an identifier which is unique e.g. a name or a number and the remaining information associated with the identifier e.g. address, telephone number, component details etc.. The identifier is stored in one array (I) and the remaining information in another (J). A label is attached to each identifier which points to the second array so that, for example, if the information is ordered only the identifier array needs to be ordered.

```
10 REM Label
20 PRINT "How many items to be stored";: INPUT No.
30 DIM I$[No.],J$[No.]
40 PRINT "Enter a unique identifier e.g. a name or part No."
50 FOR P=1 TO No.
60 INPUT "Identifier";I$[P]
70 I$[P]=I$[P]+"*"+STR$(P)
80 INPUT "Remaining details";J$[P]
90 NEXT
1000 REM A search will produce a value V which holds the ide
ntifier information"
1010 PRINT I$[V],J$[VAL(MID$(I$[V],INSTR(I$[V],"*")+1))]
```

37-41. LOOPS

Loops are a very powerful way of generating and handling data and these examples illustrate more complicated looping operations than the single or nested loop.

In the 'SPLIT LOOP', when A exceeds 4 the looping back point changes to the second FOR statement.

Within 'MIXED LOOP', there are two incrementing operations in action i.e. FOR A=1 TO 7 and A=A+1 so that instead of 7x4 pairs of values, we get 8. The +(A=4) term is to adjust at the changeover point.

In 'RANDOM LOOP' the B value skips as it exceeds the various 'greater than' conditions.

In 'CIRCULAR LOOP' the MOD function is used to change the last entry into the first so that the loop can be used to perform cyclical operations. Examples of this would be moving all the entries in an array by a given number of places using the zero position as the temporary store or plotting points to give a closed figure.

There are occasions when a set of nested loops is required but the number depends on some variable in the program. To solve this, a subscripted variable would have to be used in the loop e.g. FOR A[P]=1 TO B[P] but this is not permitted in the syntax of Amstrad. It can, however, be overcome by simulating the operations carried out during looping but without using a main FOR-----NEXT statement. Examples of this construction can be found in 'PERMUTE', 'ANAGRAM' and '2-3 TREES'.

In the example given, which is the preparation of a multiplication look-up table for P*Q (P and Q varying from 1 TO 10), a variable size loop structure is compared with a double loop. The former can be easily increased to a P*Q*R system but the simple nested loop cannot be without writing an extra loop in the program in the variable R.

37. SPLIT LOOP

```
10 REM Split loop
20 FOR A=1 TO 5: IF A>4 THEN FOR A=13 TO 16: PRINT A: NEXT:
GOTO 40
30 PRINT A
40 NEXT
```

This selects values of A equal to 1,2,3,4,13,14,15,16,

38. MIXED LOOP

```
10 REM Mixed loop
20 FOR A=1 TO 7: FOR B=5 TO 8
30 PRINT TAB(18);A;CHR$(8);",";B
40 A=A+1+(A=4): NEXT B,A
```

This selects pairs of values as follows

1,5; 2,6; 3,7; 4,8; 5,5; 6,6; 7,7; 8,8

39. RANDOM LOOP

```
10 REM 'Random' loop
20 FOR A=1 TO 4: FOR B=1 TO 2
30 IF B>1 THEN FOR B=7 TO 9: IF B>8 THEN FOR B=11 TO 15 STEP
 2: PRINT B;: NEXT: GOTO 40 ELSE PRINT B;: GOTO 40 ELSE PRIN
T B;: GOTO 50
40 NEXT
50 NEXT: PRINT
60 NEXT
```

This PRINTs 1,7,8,11,13,15, four times.

40. CIRCULAR LOOP

This loop selects adjacent points in an array so that when PLOTted, for example it will produce a closed figure. (See 'ANGLESORT').

```
10 REM Circular loop
20 INPUT N
30 DIM A[N],B[N]
40 FOR P=1 TO N: A[P]=P: B[P]=P: NEXT
50 GOSUB 1000: END
1000 FOR P=0 TO N
1010 A[P]=A[(1+P)MOD(N+1)]
1020 NEXT
1030 FOR P=1 TO N: PRINT "Join";B[P];"to";A[P]: NEXT
1040 RETURN
```

41. VARIABLE SIZE NESTED LOOP

```
10 REM Variable size nested loop
20 INPUT N: REM Number of loops
30 DIM A[N]: REM Loop variables
40 DIM B[N]: REM Loop end values
50 FOR P=1 TO N
60 A[P]=1: REM Initialise variables
70 B[P]=10: REM Initialise loop end values
80 NEXT P
90 FOR P=1 TO N
100 IF A[P]>B[P] THEN A[P]=1: REM Re-initialise inner loop v
ariables if required
110 NEXT P: Z=1
120 FOR P=1 TO N
130 Z=Z*A[P]: NEXT P: REM Calculate product
140 PRINT Z;
150 FOR X=N TO 1 STEP -1: REM Control nesting operation
160 A[X]=A[X]+1: REM Increment variable
170 IF A[X]>B[X] THEN PRINT: NEXT X: PRINT: END: REM Change
to the next loop?
180 GOTO 90: REM Next cycle of this loop
```

Without the REM statements the routine is as follows

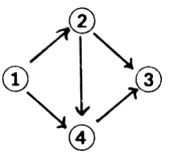
```
10 REM Variable size nested loop
20 INPUT N: DIM A[N],B[N]
30 FOR P=1 TO N: A[P]=1: B[P]=10: NEXT
40 FOR P=1 TO N: IF A[P]>B[P] THEN A[P]=1
50 NEXT: Z=1
60 FOR P=1 TO N: Z=Z*A[P]: NEXT: PRINT Z;
70 FOR X=N TO 1 STEP -1: A[X]=A[X]+1
80 IF A[X]>B[X] THEN PRINT: NEXT ELSE 40: END
```

The subroutine is of great value in problems involving permutations of the variables.

42–48. MATRICES

The matrices can be used to store information about paths between places on maps or graphs and between corners of a polygon or a solid polyhedron. For example, the matrix used in 'Universal Rotation' uses "1" to indicate which corners of a bipyramid are joined by an edge and "0" where no join exists.

The points 1 to 4 can be joined by several paths as drawn.



The path matrix would then be

$$P=\begin{bmatrix}0&1&0&1\\0&0&1&1\\0&0&0&0\\0&0&1&0\end{bmatrix}$$

P^2 has the property of representing paths made of two separate parts.

Hence

$$P^2=\begin{bmatrix}0&1&0&1\\0&0&1&1\\0&0&0&0\\0&0&1&0\end{bmatrix}*\begin{bmatrix}0&1&0&1\\0&0&1&1\\0&0&0&0\\0&0&1&0\end{bmatrix}=\begin{bmatrix}0&0&2&1\\0&0&1&0\\0&0&0&0\\0&0&0&0\end{bmatrix}$$

The 2 represents 1→3 as 1→2→3 and 1→4→3
The 1's represent 1→4 as 1→2→4 and 2→3 as 2→4→3

$$P^3=\begin{bmatrix}0&0&1&0\\0&0&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}$$

i.e. 1→2→4→3 the only triple path.

P^4 is a null matrix as there are no four part paths

Reference to 'Simultaneous Equations', 'Determinants' and 'Regression' will show other applications of matrices.

Matrix multiplication is row by column so that for example

$$\begin{bmatrix} A & B \\ E & F \end{bmatrix} * \begin{bmatrix} C & D \\ G & H \end{bmatrix} = \begin{bmatrix} A*C+B*G & A*D+B*H \\ E*C+F*G & E*D+F*H \end{bmatrix}$$

Hence, an M x N matrix can only be multiplied by an N x R matrix to give an M x R result or an A x B x C can be multiplied by a C x D to give an A x B x D answer. In each case two dimensions have to be the same and disappear from the resultant product. The order of multiplication is therefore important and in general A * B does not equal B * A.

The following example illustrates matrix multiplication using salary costs but it could equally be stock levels and values at different sites or exam results in different subjects in different years etc.

The numbers employed in each grade at six establishments are as follows:–

ESTABLISHMENT	GRADE				
	1	2	3	4	5
A	200	50	10	1	0
B	250	60	6	2	0
C	500	101	12	3	0
D	35	5	1	0	1
E	1010	190	27	2	1
F	1250	200	45	3	1

and the salary scales are:–

GRADE	£/a
1	5500
2	7500
3	9000
4	11000
5	15000

Then

$$\begin{bmatrix} 200 & 50 & 10 & 1 & 0 \\ 250 & 60 & 6 & 2 & 0 \\ 500 & 101 & 12 & 3 & 0 \\ 35 & 5 & 1 & 0 & 1 \\ 1010 & 190 & 27 & 2 & 1 \\ 1250 & 200 & 45 & 3 & 1 \end{bmatrix} * \begin{bmatrix} 5500 \\ 7500 \\ 9000 \\ 11000 \\ 15000 \end{bmatrix} = \begin{bmatrix} 1576000 \\ 1901000 \\ 3648500 \\ 254000 \\ 7260000 \\ 8828000 \end{bmatrix}$$

represented symbolically by

A[M,N]*B[N,R]=C[M,R]

gives the total salary bill at each of the six establishments.

A unit matrix is the equivalent of 1 in ordinary numbers and consists of a diagonal line of ones, the remaining entries being zeros. The product of a matrix with its inverse is a unit matrix. 'Universal Rotation' contains an example of a 3x3 unit matrix which is set up initially to build up the final transformation matrix.

Complex matrices are really associated pairs of matrices which hold the real and imaginary parts of the array of values. They have a variety of uses and a simple example is that of rotation around the origin. The expression

$(x+iy)\exp(i\theta)$

rotates the point (x,y) by an angle θ around the origin.

As $\exp(i\theta)=\cos\theta+i\sin\theta$ we have with multiple points and angles, the product of two matrices viz:–

$$\begin{bmatrix} X1+iY1 \\ X2+iY2 \\ X3+iY3 \\ X4+iY4 \end{bmatrix} * \begin{bmatrix} \cos\theta+i\sin\theta & \cos\phi+i\sin\phi & \cos\gamma+i\sin\gamma \end{bmatrix}$$

The inverse of a matrix is required in the solution of simultaneous equations and for testing whether a determinant is zero or not. The analogous inverse of a complex matrix pair is also given along with the complex conjugate which is occasionally required

42. MATRIX MULTIPLICATION

```
10 REM Matrix Multiplication
20 DEFINT J,K,L,M,N,R
30 DEF FN N$(N)=MID$(STR$(N),2)
40 INPUT "Number of rows in matrix A";M
50 INPUT "Number of columns in matrix A";N
60 INPUT "Number of columns in matrix B";R
70 DIM A[M,N],B[N,R],C[M,R]
80 FOR J=1 TO M: FOR K=1 TO N
90 PRINT "A(";FN N$(J);",";FN N$(K);")=";: INPUT A[J,K]: NEX
T K,J
100 FOR J=1 TO N: FOR K=1 TO R
110 PRINT "B(";FN N$(J);",";FN N$(K);")=";: INPUT B[J,K]: NE
XT K,J
120 REM Or use DATA INPUT (Matrices) twice to INPUT A and B)
130 GOSUB 1000: CLS: GOSUB 2000: END
1000 FOR J=1 TO M: FOR L=1 TO R
1010 FOR K=1 TO N: C[J,L]=C[J,L]+A[J,K]*B[K,L]
1020 NEXT K,L,J
1030 RETURN
2000 FOR J=1 TO M: FOR K=1 TO R
2010 LOCATE 4*K-3,2*J+3: PRINT C[J,K]
2020 NEXT K,J
2030 RETURN
2040 REM Or use PRINTOUT for MATRIX
```

EXAMPLE

```
run
Number of Rows in Matrix A? 3
Number of Columns in Matrix A? 4
Number of Columns in Matrix B? 2
A(1,1)=? 1
A(1,2)=? 2
A(1,3)=? 3
A(1,4)=? 4
A(2,1)=? 5
A(2,2)=? 6
A(2,3)=? 7
A(2,4)=? 8
A(3,1)=? 9
A(3,2)=? 10
A(3,3)=? 11
A(3,4)=? 12
B(1,1)=? 2
B(1,2)=? 4
B(2,1)=? 6
B(2,2)=? 8
B(3,1)=? 10
B(3,2)=? 12
B(4,1)=? 14
B(4,2)=? 16

 100  120
 228  280
 356  440
Ready
```

43. COMPLEX MATRIX MULTIPLICATION

```
10 REM Complex Matrix Multiplication
20 DEFINT J,K,L,M,N,P,R
30 DEF FN N$(N)=MID$(STR$(N),2)
40 INPUT "Number of rows in matrix A";M
50 INPUT "Number of columns in matrix A";N
60 INPUT "Number of columns in matrix B";R
70 CLS: P=1
80 DIM A[M,N],B[N,R],C[M,R],X[M,N],Y[N,R],Z[M,R]
90 FOR J=1 TO M: FOR K=1 TO N
100 P=P+1: LOCATE 1,P: PRINT "A(";FN N$(J);",";FN N$(K);")="
;: INPUT A[J,K]
110 LOCATE 20,P: PRINT "+i*X(";FN N$(J);",";FN N$(K);")=";:
INPUT X[J,K]: NEXT K,J
120 FOR J=1 TO N: FOR K=1 TO R
130 P=P+1: LOCATE 1,P: PRINT "B("; FN N$(J);",";FN N$(K);")=
";: INPUT B[J,K]
140 LOCATE 16,P: PRINT "i*Y(=";FN N$(J);",";FN N$(K);")=";:
INPUT Y[J,K]
150 NEXT K: NEXT J
160 REM Or use DATA INPUT (Matrices) twice to INPUT A and B)
180 GOSUB 1000: CLS: GOSUB 2000: END
1000 FOR J=1 TO M: FOR L=1 TO R: FOR K=1 TO N
1010 C[J,L]=C[J,L]+A[J,K]*B[K,L]-X[J,K]*Y[K,L]
1020 Z[J,L]=Z[J,L]+A[J,K]*Y[K,L]+X[J,K]*B[K,L]
1030 NEXT K,L,J
1040 RETURN
2000 FOR J=1 TO M: FOR K=1 TO N
2010 LOCATE 8*K-3,2*J+3: PRINT C[J,K];"+i*";Z[J,K]
2020 NEXT K,J
2030 RETURN
2040 REM Or use PRINTOUT for MATRIX
```

44. MATRIX INVERSION

You cannot divide two matrices but can derive A^{-1} and multiply using the matrix multiplication rules provided that A is square and its determinant not zero.

This derivation of A^{-1} is based on the Gauss–Jordan method of progressive substitution whereby three loops are established and the individual entries are operated on in turn and the values in the matrix immediately replaced by the new value. Thus in the case of 3x3 matrix, there are 27 operations so that the matrix gradually changes into its inverse in 27 stages. Relating this to equations, if

$$Y_1 = a_{11} X_1 + a_{12} X_2 + a_{13} X_3$$
$$Y_2 = a_{21} X_1 + a_{22} X_2 + a_{23} X_3$$
$$Y_3 = a_{31} X_1 + a_{32} X_2 + a_{33} X_3$$

for example. Then solving the first equation for X_1 and substituting gives (if $a_{11} <> 0$)

$$X_1 = A_{11} Y_1 + A_{12} X_2 + A_{13} X_3$$
$$Y_2 = A_{21} Y_1 + A_{22} X_2 + A_{23} X_3$$
$$Y_3 = A_{31} Y_1 + A_{32} X_2 + A_{33} X_3$$

where

$A_{11} = 1/a_{11}$ $\quad A_{12} = -a_{12}/a_{11}$ $\quad A_{13} = -a_{13}/a_{11}$

$A_{21} = a_{21}/a_{11}$ $\quad A_{22} = a_{22} - a_{21} a_{12}/a_{11}$ $\quad A_{23} = a_{23} - a_{21} a_{13}/a_{11}$

$A_{31} = a_{31}/a_{11}$ $\quad A_{32} = a_{32} - a_{31} a_{12}/a_{11}$ $\quad A_{33} = a_{33} - a_{31} a_{13}/a_{11}$

Repeating this process through the three cycles produces the inverse.

SUB 1120 checks for zeros in the leading diagonal and exchanges two columns if possible to avoid a 'Division by zero' problem at the beginning of the I loop. Array M remembers which columns have been interchanged and in SUB 1160 exchanges the same rows in the inverse to compensate.

If two rows (or columns) are identical or one a multiple of the other or if a line of zeros is present, then the matrix cannot be inverted as its determinant is zero. This is picked up by the IF statement at the beginning of the routine.

The IF J<>N+(I=N) statement in the K loop dispenses with the need for a second set of loops found in some versions of this routine.

```
10 REM Matrix Inversion
20 DEFINT I,J,K,N,T: DEF FN N$(N)=MID$(STR$(N),2): N=0
30 WHILE N<=0: INPUT "N=";N: WEND
40 DIM A(N,N): FOR I=1 TO N: FOR J=1 TO N
50 PRINT "A(";FN N$(I);",";FN N$(J);")=";: INPUT A(I,J): NEX
T J,I
60 GOSUB 1000: GOSUB 2000: END
1000 DIM M(N,N): T=0: GOSUB 1120
1010 FOR I=1 TO N: IF ABS(A(I,I))<0.0000001 THEN T=1: RETURN
1020 A(I,I)=1/A(I,I)
1030 FOR J=1 TO N: IF J=I THEN 1100
1040 A(J,I)=A(J,I)*A(I,I)
1050 FOR K=1 TO N: IF K=I THEN 1090
1060 A(J,K)=A(J,K)-A(J,I)*A(I,K)
1070 IF J<>N+(I=N) THEN 1090
1080 A(I,K)=-A(I,I)*A(I,K)
1090 NEXT K
1100 NEXT J,I: IF T THEN GOSUB 1160
1110 RETURN
1120 FOR I=1 TO N: IF A(I,I)=0 THEN T=-1: GOSUB 1140
1130 NEXT I: RETURN
1140 FOR J=I TO N: IF A(I,J)<>0 THEN FOR K=1 TO N: SW=A(K,I)
: A(K,I)=A(K,J): A(K,J)=SW: NEXT K: M(I,J)=1: RETURN
1150 NEXT J: RETURN
1160 FOR I=1 TO N: FOR J=1 TO N
1170 IF M(I,J)=1 THEN FOR K=1 TO N: SW=A(I,K): A(I,K)=A(J,K)
: A(J,K)=SW: NEXT K
1180 NEXT J,I: RETURN
2000 ERASE M: IF T<1 THEN FOR I=1 TO N: FOR J=1 TO N: PRINT
"A(";FN N$(I);",";FN N$(J);")=";A(I,J): NEXT J,I: RETURN
2010 PRINT "Singular Matrix, there is no inverse": RETURN
```

EXAMPLE

$$\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \\ 1 & 0 & 3 \end{bmatrix} \qquad \begin{bmatrix} -1.5 & 1.125 & .25 \\ 0 & .25 & -.5 \\ .5 & -.375 & .25 \end{bmatrix}$$

45. COMPLEX MATRIX INVERSION

```
10 REM Complex matrix inversion
20 DEFINT A,B,N,J,K,T
30 DEF FN M(A,B)=R[A,B]*R[A,B]+I[A,B]*I[A,B]
40 DEF FN Z$(A$,A,BOOLE)=MID$(A$,1+A,-LEN(A$)*BOOLE)
50 DEF FN N$(N)=MID$(STR$(N),2)
60 INPUT "Enter the rank of the matrix";N$
70 IF N$="" THEN 60
80 T=1: IF LEFT$(N$,1)="+" THEN T=2
90 FOR P=T TO LEN(N$): IF MID$(N$,P)<"0" OR MID$(N$,P)>"9" T
HEN 60
100 NEXT: N=VAL(N$):  IF N<=0 THEN 60
110 DIM R[N,N],I[N,N]
120 PRINT "   Now enter the individual terms in thematrix wh
ich can be real, imaginary or  complex. Typically, they will
 be of the form a+ib where i=SQR(-1). Type in a andb. Press
RETURN to continue": INPUT K$
130 CLS: T=1
140 FOR I=1 TO N: FOR J=1 TO N: T=T+1
150 LOCATE 1,T: PRINT "a";FN N$(I);FN N$(J);" is ";: INPUT R
(I,J)
160 LOCATE 20,T: PRINT "+i*";;: INPUT I(I,J)
170 NEXT J,I
180 GOSUB 1000: GOSUB 2000: END
1000 DIM M[N,N]: T=0: GOSUB 1120
1010 FOR I=1 TO N: Z=FN M(I,I): IF ABS(Z)<0.00000001 THEN T=
1: GOTO 2000
1020 R[I,I]=R[I,I]/Z: I[I,I]=-I[I,I]/Z
1030 FOR J=1 TO N: IF J=I THEN 1100
1040 Z=R[J,I]: R[J,I]=Z*R[I,I]-I[J,I]*I[I,I]: I[J,I]=I[J,I]*
R[I,I]+Z*I[I,I]
1050 FOR K=1 TO N: IF K=I THEN 1090
1060 R[J,K]=R[J,K]-R[J,I]*R[I,K]+I[J,I]*I[I,K]: I[J,K]=I[J,K
]-I[J,I]*R[I,K]-R[J,I]*I[I,K]
1070 IF J<>N+(I=N) THEN 1090
1080 Z=R[I,K]: R[I,K]=-R[I,I]*Z+I[I,I]*I[I,K]: I[I,K]=-I[I,I
]*Z-R[I,I]*I[I,K]
1090 NEXT
1100 NEXT J,I: IF T THEN GOSUB 1160
1110 RETURN
1120 FOR I=1 TO N: IF FN M(I,I)=0 THEN T=-1: GOSUB 1140
1130 NEXT: RETURN
1140 FOR J=1 TO N: IF FN M(I,J)<>0 THEN FOR K=1 TO N: SW=R[K
,I]: R[K,I]=R[K,J]: R[K,J]=SW: SW=I[K,I]: I[K,I]=I[K,J]: I[K
,J]=SW: M[I,J]=1: NEXT: RETURN
1150 NEXT: RETURN
1160 FOR I=1 TO N: FOR J=1 TO N:
1170 IF M[I,J]=1 THEN FOR K=1 TO N: SW=R[I,K]: R[I,K]=R[J,K]
: R[J,K]=SW: SW=I[I,K]: I[I,K]=I[J,K]: I[J,K]=SW: NEXT
1180 NEXT J,I: RETURN
1190 RETURN
```

```
2000 ERASE M: IF T>0 THEN PRINT "Singular matrix, no inverse
.": RETURN
2005 PRINT "The inverse matrix is": PRINT
2010 FOR I=1 TO N: FOR J=1 TO N: GOSUB 2050
2020 PRINT "b(";FN N$(I);",";FN N$(J);")=";Z$
2030 NEXT J,I
2040 RETURN
2050 X$=FN Z$("-",0,R[I,J]<0)+FN Z$(STR$(R[I,J]),1,R[I,J]<>0
)+FN Z$("+",0,ABS(R[I,J])>0 AND I[I,J]>0)+FN Z$("-",0,I[I,J]
<0)
2060 Y$=FN Z$(STR$(I[I,J]),1,I[I,J]<>0 AND ABS(I[I,J])<>1)+F
N Z$("i",0,I[I,J]<>0)+FN Z$("0",0,ABS(R[I,J])<0.000001 AND A
BS(I[I,J])<0.000001)
2070 Z$=X$+Y$
2080 RETURN
```

EXAMPLE

```
Enter the rank of the matrix? 3
Now enter the individual terms in the
matrix which can be real imaginary or
complex. Generally, they will be of
the form a+ib where i=SQR(-1). Type
in a and b. Press RETURN to continue

a11 is ?              1 +i*? 0
a12 is ?              0 +i*? 0
a13 is ?              0 +i*? 5
a21 is ?              0 +i*? -2
a22 is ?              2 +i*? 0
a23 is ?              0 +i*? 0
a31 is ?              1 +i*? 0
a32 is ?              1 +i*? 1
a33 is ?              0 +i*? 0



B11 = 0
B12 = -.5+.5i
B13 = -i
B21 = 0
B22 = -.5i
B23 = 1
B31 = -.2i
B32 = -.1-.1i
B33 = .2
```

46. UNIT MATRIX

DIM is used to create the zeros and the loop the diagonal row of ones.

```
10 REM Unit matrix
20 DEFINT I,J,N
30 INPUT "Rank of the matrix";N: IF N<1 THEN 30
40 GOSUB 1000: GOSUB 2000: END
1000 DIM A[N,N]
1010 FOR I=1 TO N: A[I,I]=1: NEXT
1020 RETURN
2000 FOR I=1 TO N: FOR J=1 TO N
2010 PRINT A[I,J];: NEXT: PRINT: NEXT
2020 RETURN
```

47. TRANSPOSE

In a transpose of a square matrix, rows become columns and vice versa

```
10 REM Transpose of a matrix
20 DEFINT J,K,N
30 INPUT "Rank of matrix";N: IF N<1 THEN 30
40 DIM A[N,N]
50 FOR J=1 TO N: FOR K=1 TO N: INPUT A[J,K]: NEXT K,J
60 GOSUB 1000: GOSUB 2000: END
1000 FOR J=1 TO N: FOR K=J TO N
1010 IF J<>K THEN SW=A[J,K]: A[J,K]=A[K,J]: A[K,J]=SW
1020 NEXT: NEXT
1030 RETURN
2000 FOR J=1 TO N: FOR K=1 TO N
2010 PRINT A[J,K];: NEXT K: PRINT: NEXT J
2020 RETURN
```

48. COMPLEX CONJUGATE

The complex conjugate has the opposite sign on the imaginary parts.

```
10 REM Complex conjugate of a complex pair of matrices
20 DEFINT J,K,N
30 INPUT "Rank of matrix";N: IF N<1 THEN 30
40 DIM REAL [N,N],IMAG [N,N]
50 FOR J=1 TO N: FOR K=1 TO N: INPUT REAL [J,K],IMAG [J,K]:
NEXT K,J: REM Enter REAL [J,K] and IMAG [J,K] as two real nu
mbers separated by a comma
60 GOSUB 1000: GOSUB 2000: END
1000 FOR J=1 TO N: FOR K=1 TO N
1010 IMAG [J,K]=-IMAG [J,K]
1020 NEXT: NEXT
1030 RETURN
2000 FOR J=1 TO N: FOR K=1 TO N
2010 PRINT REAL [J,K];IMAG [J,K];"i": NEXT K,J
2020 RETURN
```

49. MENU

The object of having a 'MENU' in a program is to help the user to interact with the computer and to be able to choose a specific part of the program without having to run all the way through the rest of it. Complicated programs often have 'Help' menus which you can return to if you get out of your depth and don't know what to do next.

The following program illustrates a simple 'Menu' arrangement using ON---GOSUB as the active element. Breaking into the program anywhere and typing GOTO (line 120) will bring the menu back to the screen. Normally the various subroutines will return to (line 120) after completion and the last subroutine will enable you to terminate the program. Other subroutines may allow data to be saved or loaded.

The point of note in the program is the POKE in the second line which alters CHR$(208) to give a better underline. M$[10] holds the chapter titles and these are printed by the P loop.

The choice is made as M$[0] which is tested in SUB 300 for numerical characteristics otherwise FAIL becomes true and the choice is requested again. Part of the complication in this area is preventing the Menu being inched up the screen by various messages so as far as possible the window is reserved for active inputs. However if you accidently ENTER a comma, you can't stop the 'Redo from start' message from appearing.

```
10 REM Menu
20 SYMBOL AFTER 208: H=HIMEM+1: POKE H+1,0
30 WINDOW #1,1,40,22,25: DIM M$[10]
40 NL$=CHR$(10)+CHR$(13)
50 M$[1]="   1.   Load Data"
60 M$[2]="   2.   Save Data"
70 M$[3]="   3.   Second Choice"
80 M$[4]="   4.   Third Choice"
90 REM etc.
100 M$[9]="   9.   Ninth Choice"
110 M$[10]="  10.   Finish"
120 CLS: LOCATE 18,2: PRINT "MENU"
130 LOCATE 18,3: FOR P=1 TO 4: PRINT CHR$(208);: NEXT: PRINT
140 FOR P=1 TO 10: LOCATE 5,4+P
150 PRINT M$[P]: NEXT: T$[0]=""
160 PRINT #1,"Type in your choice 1-10"
170 T$=INKEY$: IF T$="" THEN 170
180 PRINT #1,T$;: IF T$<>CHR$(13) THEN T$[0]=T$[0]+T$: GOTO
170
190 GOSUB 300: IF NOT FAIL THEN M%=VAL(T$[0]) ELSE 170
200 IF M%>10 OR M%<1 THEN 170
```

```
210 ON M% GOSUB 400,500,600,700,800,900,1000,1100,1200,1310:
 REM Or as appropriate
220 GOTO 120
240 END
300 FAIL=0
310 FOR P=1 TO LEN(M$[O]): Z$=MID$(M$[O],P,1)
320 IF Z$<"0" OR Z$>"9" THEN FAIL=-1
330 NEXT: RETURN
400 REM Load the data from the Datacorder
410 CLS: PRINT "   Is the Datacorder ready to INPUT the stor
ed data (y/n)?";NL$
420 IF INKEY$<>"Y" AND INKEY$<>"y" THEN 420
430 OPENIN D$
440 INPUT #9,A,B,C: REM Here A,B,C means the variables A,B,C
 whose values you have stored previously
450 CLOSEIN
460 RETURN
500 REM Save Data
510 CLS: PRINT "Is the tape recorder ready to record the dat
a. (y/n)";NL$;"  Make a note of the position and which tape"
520 IF INKEY$<>"y" AND INKEY$<>"Y" THEN 520
530 PRINT "Type in the name of the file";: INPUT D$
540 OPENOUT D$
550 PRINT #9,A,B,C: REM Here A,B,C means the names of the va
riables whose values you wish to save. See '2-3 TREE' for an
 example
560 CLOSEOUT
570 RETURN
1300 REM Finish
1310 CLS: PRINT "Do you really want to finish?";NL$
1320 PRINT "Press ";CHR$(34);"Y";CHR$(34);" for finish else
<ENTER>";: INPUT K$
1330 K$=UPPER$(K$): IF K$="Y" THEN NEW ELSE RETURN
```

50. MERGE

This subroutine merges two lists, each of which is already in order, into a single ordered list. If you need to add extra values to an ordered list it is much more efficient to sort them first and then merge the two lists. Slotting them in individually in the correct place or adding them to the end of the list and resorting takes a lot longer.

MERGE is used in MERGESORT and MIN/MAX

```
10 REM Merge
20 PRINT "Length of lists";"REM Enter as two numbers separat
ed by a comma"
25 DEF FN A$(A)=MID$(STR$(A),2)
30 INPUT M,N: IF M=0 OR N=0 THEN 30
40 M=M+1: N=N+1: TITCH=0
50 DIM A[M],B[N],C[M+N]
60 FOR P=2 TO M: PRINT "A(";FN A$(P-1);")=";: INPUT A[P]
70 IF A[P]<TITCH THEN TITCH=A[P]
80 NEXT
90 FOR P=2 TO N: PRINT "B(";FN A$(P-1);")=";: INPUT B[P]
100 IF B[P]<TITCH THEN TITCH=B[P]
110 NEXT
120 GOSUB 1000: GOSUB 2000: END
1000 A[1]=TITCH-1: B[1]=A[1]
1010 Y=M: Z=N
1020 WHILE A[Y]>B[Z]: C[Y+Z]=A[Y]: IF Y>1 THEN Y=Y-1: WEND
1030 C[Y+Z]=B[Z]: IF Z>1 THEN Z=Z-1: GOTO 1020
1040 RETURN
2000 FOR P=3 TO M+N: PRINT C[P];: NEXT
2010 ERASE A,B,C: RETURN
```

51. MIN/MAX

This routine gives the same answer as MIN and MAX but is much more flexible. The subroutine finds the maximum and minimum values in a list. It works by ordering adjacent entered values and selecting the maximum and minimum in each pair. These values are then compared in 4's, 8's, 16's, etc. selecting the maximum and minimum in each case.

```
10 REM Min/Max
20 DEFINT A,B,N,P,S: N=0
30 DEF FN A$(A)=MID$(STR$(A),2)
40 WHILE N<=0: INPUT "Number of values";N: WEND
50 GOSUB 1000: GOSUB 2000: END
1000 M=N: P=0
1010 WHILE M>1: M=M/2: P=P+1: WEND
1020 S=2^P: DIM A[S]
1030 FOR A=1 TO N: PRINT "A(";FN A$(A);")=";: INPUT A[A]
1040 A=A+1: IF A>N THEN 1080
1050 PRINT "A(";FN A$(A);")=";: INPUT A[A]
1060 IF A[A]>A[A-1] THEN SW=A[A]: A[A]=A[A-1]: A[A-1]=SW
1070 NEXT
1080 IF M<>1 THEN FOR A=N+1 TO S: A[A]=A[N]: NEXT
1090 M=2
1100 FOR B=0 TO S/2/M-1: T=2*B*M
1110 IF A[1+T]<A[1+T+M] THEN A[1+T]=A[1+T+M]
1120 IF A[T+M]<A[T+M+M] THEN A[T+M+M]=A[T+M]
1130 NEXT: M=M+M: IF M<S THEN 1100
1140 RETURN
2000 PRINT: PRINT "Min=";;A[S];SPACE$(5);"Max=";A[1]
```

52. MINIMUM, MAXIMUM, MEAN, MEDIAN AND MODE

By putting the numbers in ascending order it is easy to find the minimum and maximum. The mean is established from the sum of the values during the input phase and the median is half way up the list. This will be one of the values if there is an odd number of them or the mean of two adjacent ones if there is an even number.

The mode is more difficult to derive as you have to count the number of entries of each value. T[0,Z] does this and T[1,Z] stores the different actual values. At E, T[0,Z] is ordered by a bubblesort into descending order and the mode corresponds to the first value in T[1,Z]. T[0,Z] and T[1,Z] are both swapped in the sorting.

This subroutine PRINTs out the above values for a list of positive numbers.

```
10 REM Minimum, Maximum, Mean, Median and Mode
20 DEF FN A$(A)=MID$(STR$(A),2)
30 INPUT "Number of values";N: IF N<=0 THEN 30
40 DIM V[N]: S=0
50 FOR P=1 TO N: PRINT "V(";FN A$(P);")=";: INPUT V[P]: S=S+
V[P]: NEXT
60 GOSUB 1000: GOSUB 2000: END
1000 Q=0: FOR P=1 TO N-1
1010 IF V[P+1]<V[P] THEN SW=V[P]: V[P]=V[P+1]: V[P+1]=SW: Q=
Q+1
1020 NEXT: IF Q<>0 THEN 1000
1030 P=1: Z=1: DIM T[1,N]
1040 IF V[P]=V[P+1] THEN T[0,Z]=T[0,Z]+1: P=P+1: IF P<N THEN
 1040
1050 T[1,Z]=V[P]: P=P+1: Z=Z+1+T[0,Z]: IF P<N THEN 1040
1060 Q=0: FOR P=1 TO N-1
1070 X=P+1: IF T[0,X]>T[0,P] THEN SW=T[0,P]: T[0,P]=T[0,X]:
T[0,X]=SW: SW=T[1,P]: T[1,P]=T[1,X]: T[1,X]=SW: Q=Q+1
1080 NEXT: IF Q<>0 THEN 1060
1090 RETURN
2000 PRINT "Minimum =";V[1]: PRINT
2010 PRINT "Maximum =";V[N]: PRINT
2020 PRINT "Mean    =";S/N: PRINT
2030 IF 2*INT(N/2)=N THEN PRINT "Median   =";(V[N/2]+V[N/2+1
])/2 ELSE PRINT "Median  =";V[N/2]
2040 PRINT: PRINT "Mode    =";T[1,1]: PRINT
2050 ERASE T: RETURN
```

53. MODULUS

Although one rarely needs a modulus sign, the routine is included as an example of using POKE to alter an ASCII character. Reference to 'DISPLAY FILE' explains the values used in POKE.

```
10 REM Modulus
20 GOSUB 1000: GOSUB 2000: END
1000 SYMBOL AFTER 209
1010 H=HIMEM+1
1020 FOR P=H TO H+7
1030 POKE P,4: POKE P+16,64
1040 NEXT
1050 RETURN
2000 PRINT CHR$(209);"M";CHR$(211);"is the modulus of M"
2010 RETURN
```

54. NAME FILTER

This subroutine allows names to be stored. The permitted characters can be capital or lower case letter, hyphens, full stops, spaces and apostrophes. Words containing other characters are rejected. The loop in P tests for the normal characters found in names but if the test fails, Q is reduced by one so as not to leave a blank in B$[N].

```
10 REM Name filter
20 INPUT "Number of names";N: IF N<1 THEN 20
30 DIM B$[N]
40 GOSUB 1000: GOSUB 2000: END
1000 FOR Q=1 TO N: PRINT "Name";Q;" is ";: INPUT A$
1010 FOR P=1 TO LEN(A$): Z$=MID$(A$,P,1)
1020 IF Z$="-" OR Z$="." OR Z$=" " OR Z$="'" OR Z$>="A" AND
Z$<="Z" OR Z$>="a" AND Z$<="z" THEN NEXT: B$[Q]=A$: GOTO 104
0
1030 PRINT A$;" is not a name": Q=Q-1
1040 NEXT Q
1050 RETURN
2000 FOR P=1 TO N: PRINT B$[P]: NEXT
2010 RETURN
```

EXAMPLE

```
run                                (Output)
Number of names? 13                      J. Lucas-Tooth
Name   1   is  ? J. Lucas-Tooth          T. Harding
Name   2   is  ? T. Harding              Mr. R. Smith
Name   3   is  ? Mr. 5. Smith            I. O'Connel
Mr. 5 Smith is not a name                Mr.P. Ward-Jones
Name   3   is  ? Mr. R. Smith            Rachelina
Name   4   is  ? I. O'Connel            Johnson W.
Name   5   is  ? Mr. P. Ward-Jones       Clive Bosworth
Name   6   is  ? Rachelina              Miss J. Robinson
Name   7   is  ? Johnson W.             T.K. Jones
Name   8   is  ? Clive Bosworth         J. Bu'Lock
Name   9   is  ? Miss J. Robinson       Alison
Name   10   is  ? T;K. Jones           William Peterson
T;K. Jones is not a name
Name   10   is  ? T.K. Jones
Name   11   is  ? J. Bu'Lock
Name   12   is  ? Alison
Name   13   is  ? William Peterson
Ready
```

55. PERMUTE

There are n! (factorial n) permutation of n items and a number of programs need the ability to generate them. The subroutine given below is used in 'ANAGRAM', 'DETERMINANTS BY LAPLACE DEVELOPMENT' and '2–3 TREES' for example.

All permutations of a set of items can be made by starting with one character and adding the next character in each possible position. This is illustrated for ABCD in the first figure. The subroutine given here works a little differently by progressively rotating the characters to the left (or right if you start at the other end). The sequence of permutations for ABCD is then as shown in the second figure. Where a sequence is restored by a second rotation the result is ignored.

The following simple program (for four characters) performs the rotate operation.

```
10 REM Permute, simple
20 INPUT A$: REM LEN(A$)=4
30 FOR R=1 TO 4
40 FOR Q=1 TO 3
50 FOR P=1 TO 2
60 PRINT A$;" ";
70 N=1: GOSUB 110: NEXT P
80 N=2: GOSUB 110: NEXT Q
90 N=3: GOSUB 110: NEXT R
100 END
110 Z$=LEFT$(A$,1)
120 FOR X=1 TO N
130 MID$(A$,X,1)=MID$(A$,X+1,1)
140 NEXT X
160 MID$(A$,X)=Z$
170 RETURN
```

This program cries out for an outer loop which would enable it to permute any number of characters but it would involve the use of a subscripted variable which is not permitted as a loop variable. Instead, the loop structure without FOR-----NEXT is used (see LOOPS) and produces the following subroutine. The variables in A[L] become the loop variables and those in B[L] the end values of each loop (2,3,4 etc.). Line 2 initialises the beginning and end values of the loops. Line 1020 reinitialises A[X] if A[X] has passed the end value B[X].

The loop in X controls which loop is operating and Z$ holds the first

character whilst P perfoms the Left Rotate routine. A[X] is incremented and tested against its end value.

The subroutine PRINTs all the permutations of the characters in a string.

```
10 REM Permute universal
20 INPUT A$: IF A$="" OR LEN(A$)=1 THEN 20
30 GOSUB 1000: END
1000 L=LEN(A$)-1: DIM A[L],B[L]
1010 FOR P=1 TO L: A[P]=1: B[P]=L-P+2: NEXT
1020 FOR P=1 TO L: IF A[P]>B[P] THEN A[P]=1
1030 NEXT
1040 PRINT A$+SPACE$(7-L);
1050 FOR X=L TO 1 STEP -1
1060 Z$=LEFT$(A$,1): FOR P=1 TO LEN(A$)-X
1070 MID$(A$,P,1)=MID$(A$,P+1,1): NEXT
1080 MID$(A$,P,1)=Z$
1090 A[X]=A[X]+1: IF A[X]>B[X] THEN NEXT: GOTO 1100 ELSE 102
0
1100 ERASE A,B: RETURN
```

EXAMPLE

First 100 Permutations of 'teaser'

teaser	etaser	eatser	aetser	ateser
taeser	easter	aester	aseter	saeter
seater	esater	asteer	sateer	staeer
tsaeer	taseer	atseer	steaer	tseaer
tesaer	etsaer	estaer	setaer	easetr
aesetr	aseetr	saeetr	seaetr	esaetr
aesetr	seeatr	eseatr	eesatr	eesatr
eseatr	seeatr	eeastr	eeastr	eaestr
aeestr	aeestr	eaestr	aseter	saeter
seater	esater	easter	aester	setaer
estaer	etsaer	tesaer	tseaer	steaer
etaser	teaser	taeser	ateser	aetser
eatser	taseer	atseer	astear	sateer
staeer	tsaeer	setear	estear	etsear
tesear	tseear	steear	etesar	teesar
teesar	etesar	eetsar	eetsar	tesear
etsear	estear	setear	steear	tseear
esetar	seetar	seetar	esetar	eestar
eestar	eteasr	teeasr	teeasr	eteasr
eetasr	eetasr	teaesr	etaesr	eatesr

56. PERMUTATIONS OF THREE NUMBERS

This is a simplified permute subroutine for just three numbers which was developed for the crystal program to permute the face indices h,k and l

As the first index can be chosen in three ways, the second in only two (as we have already chosen one) and the third is then fixed, the total number of ways is 3*2*1=6. (n! for n values)

In the A loop the choice begins with

h k l

Two terms are interchanged depending on the value of T which is either 2 (A even) or 1 (A odd) and so the sequence of changes becomes

A	A[0]	A[1]	A[2]	
0	h	k	l	interchange 0 and 2
1	l	k	h	interchange 0 and 1
2	k	l	h	interchange 0 and 2
3	h	l	k	interchange 0 and 1
4	l	h	k	interchange 0 and 2
5	k	h	l	interchange 0 and 1
6	h	k	l	

The subroutine produces the six permutations of any three numbers h, k and l.

```
10 REM Permutations of three numbers
20 DIM A[2],P[6,2]
30 INPUT "h=";A[0]
40 INPUT "k=";A[1]
50 INPUT "l=";A[2]
60 GOSUB 1000: GOSUB 2000: END
1000 FOR A=0 TO 6: T=2-(A/2-INT(A/2))*2
1010 FOR B=0 TO 2: P[A,B]=A[B]: NEXT
1020 SW=A[0]: A[0]=A[T]: A[T]=SW: NEXT
1030 RETURN
2000 FOR A=0 TO 6: FOR B=0 TO 2
2010 PRINT P[A,B];",";: NEXT: PRINT CHR$(8);CHR$(32);
2020 PRINT: NEXT
2030 RETURN
```

57. POSTWAR INFLATION

The data for this simple calculation are taken from the London and Cambridge Economic Bulletin Index and, from 1974, from the Department of Employment Gazette, Monthly, Table 6.1

The routine shows the value of money between 1945 and 1985 and its effect on prices.

```
10 REM Postwar inflation
20 DATA 1.00,.945,.9125,.8458,.8209,.8,.7333,.6708,.65,.6375
,.6125,.5833,.5625,.5458,.5417,.5375,.5208,.4958,.4917,.4708
,.4417,.4333
30 DATA .425,.4083,.3875,.3583,.33,.31,.28,.2158,.1799,.1458
,.1252,.1139,.1041,.088,.0778,.0695,.0662,.0630,.0559
40 DEFREAL I: DEFINT P: DEFSTR A,V,Y
50 DIM I[41]
60 FOR P=1 TO 41: READ I[P]: NEXT
70 PRINT "Enter the value and the year of purchase(1945-1985
)": PRINT: PRINT "£";;: LINE INPUT V: LOCATE POS(#0)+LEN(V)+
1,VPOS(#0)-1: PRINT " in 19";: LINE INPUT Y
80 PRINT: PRINT "In what year would you like to know its val
ue? 19";: LINE INPUT A
90 GOSUB 2000: END
2000 PRINT: PRINT "It cost #";V;" in 19";Y
2010 V!=VAL(V)
2020 PRINT: PRINT "Its value in 19";A;" was £"; USING "#####
.##";V!*I[VAL(A)-44]/I[VAL(Y)-44]
2030 PRINT: PRINT "It would cost £"; USING "#####.##";V!*I[V
AL(Y)-44)/I[VAL(A)-44];: PRINT " to buy in 19";A
2040 PRINT: PRINT "The factor is "; USING "##.####";I[VAL(Y)
-44]/I[VAL(A)-44];: PRINT " to one"
2050 RETURN
```

58. PRIME NUMBERS

Two routines are given, one is faster than the other but requires more memory.

The first routine finds the primes by eliminating any number which has a factor. Taking 1, 2 and 3 as primes without calculation the program adds 2 on and tries to find if 5 has any factors. As it fails to find a factor, 5 is a prime number and it moves on to 7 and so on. Divisions for factors are taken up to approximately the square root of the number (line 2060) as there cannot be a larger factor without you having already found a factor less than the square root.

A variant of this routine stores the primes but takes up less memory than the third subroutine which is faster and uses Eratosthenes' Sieve.

This technique establishes an array containing all the odd numbers from 3 up to n. The smallest number only is allowed through the sieve and all multiples of it are eliminated from the list. The smallest number left is then let through and its multiples eliminated and so on.

The Table gives a comparison of the times and memory requirements of the two routines.

This subroutine generates prime numbers up to N

```
10 REM Primes by division
20 INPUT "N=";N: N=INT(ABS(N))
30 GOSUB 2000: END
2000 A=3
2010 IF N>=1 THEN PRINT 1
2020 IF N>=2 THEN PRINT 2
2030 IF N>=3 THEN PRINT A
2040 B=3: A=A+2
2050 IF A-B*INT(A/B)=0 THEN 2040
2060 IF B>=INT(A/B) THEN 2080
2070 B=B+2: GOTO 2050
2080 IF A<=N THEN PRINT A
2090 IF A>=N THEN RETURN ELSE 2040
```

This subroutine generates and stores prime numbers up to SQR(N) and calculates the number of them as well.

```
10 REM Primes up to root N with storage
20 INPUT "N=";N
30 N=INT(ABS(N)): RT=SQR(N)
40 GOSUB 1000: GOSUB 2000: END
1000 A=3: Z=4: DIM N[RT/2+2]
1010 IF N>=1 THEN N[1]=1
1020 IF N>=4 THEN N[2]=2
1030 IF N>=9 THEN N[3]=3
1040 B=3: A=A+2
1050 IF A-B*INT(A/B)=0 THEN 1040
1060 IF B>=INT(A/B) THEN 1080
1070 B=B+2: GOTO 1050
1080 IF A<=RT THEN N[Z]=A: Z=Z+1
1090 IF A>=RT THEN RETURN ELSE 1040
2000 FOR P=1 TO RT/2+2: IF N[P]<>0 THEN PRINT N[P]
2010 NEXT
2020 PRINT "Number of prime numbers up to SQR";N;"(";RT;") i
s";Z-1
2030 ERASE N: RETURN
```

```
10 REM Eratosthenes' Sieve
20 DEFINT N,P
30 PRINT "Maximum number (>3) to be considered": INPUT N
40 IF N<=2 THEN 30
50 GOSUB 1000: GOSUB 2000: END
1000 N=INT((N+1)/2): DIM A[N]: A[1]=3
1010 FOR P=1 TO N-1: A[P+1]=A[P]+2: NEXT
1020 FOR P=1 TO N: IF A[P]<>0 THEN GOSUB 1050
1030 NEXT : CLS
1040 RETURN
1050 FOR Q=P TO N-A[P] STEP A[P]: IF Q+A[P]<=N THEN A[Q+A[P]
]=0
1060 NEXT : RETURN
2000 PRINT 1: PRINT 2
2010 FOR P=1 TO N-1: IF A[P]<>0 THEN PRINT A[P]
2020 NEXT: RETURN
```

Comparison of Time and Memory Used.

Number	*Divison for Factors*		*Eratosthenes' Sieve*	
	Time (Secs)	*Memory*	*Time (secs)*	*Memory*
10	.27	720	.34	842
20	.55	720	.55	867
50	.96	720	1.05	942
100	2.45	720	1.89	1067
200	5.91	720	3.99	1317
500	17.12	720	9.99	2067
1000	39.81	720	19.55	3317
2000	97.81	720	39.81	5817
5000	325.99	720	99.55	13317

59. PRINTOUT FOR A MATRIX OR A DETERMINANT

It may be necessary to round to four significant figures to prevent overprinting in large matrices if the values are not integers.

The subroutine displays arrays up to nine rows and twelve columns with matrix or determinant brackets.

```
10 REM Printout for matrix or determinant
15 CLS: INPUT "Mode";S
20 INPUT "M=";M: INPUT "N=";N: IF M<1 OR M<1 THEN 20
30 DIM A[M,N]: FOR J=1 TO M: FOR K=1 TO N
40 INPUT A[J,K]: NEXT K,J
50 GOSUB 2000: END
2000 MODE S: TAG
2010 MOVE 120,390: PRINT "The matrix is";: REM Or "The deter
minant is"
2020 PLOT 120,374: DRAWR 208/S,0: REM Or PLOT 120,374: DRAWR
 288/S,0
2030 FOR J=1 TO M: FOR K=1 TO N
2040 MOVE 96*K-40,350-32*J: PRINT A(J,K);: NEXT K,J
2050 PLOT 72,326: DRAWR -20,0: DRAWR 0,4-32*M: DRAWR 20,0
2060 PLOT 96*N+64/S,326: DRAWR 20,0: DRAWR 0,4-32*M: DRAWR -
20,0
2070 TAGOFF: RETURN
2080 REM For determinants use PLOT 52,326: DRAWR 0,4-32*M an
d PLOT 96*N+84/S,326: DRAWR 0,4-32*M in lines 2050 and 2060
respectively
```

60. PROJECTION

The 3–D origin is assumed to be near the centre of the TV screen at (320,196) and the projection relationship is

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} -\cos\alpha & \cos\beta & 0 \\ -\sin\alpha & -\sin\beta & 1 \\ 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} 320 \\ 196 \\ 0 \end{bmatrix}$$

Taking the aspect ratio of the screen as 0.92*, the lengths of the axes are

Oz=162 pixels
Oy=176.2 pixels
Ox=61.39 pixels

and the angles are given by

$\tan\alpha = 1/3$
$\tan\beta = 1/27$

The relationship is then

X=176.2yCOS(ARCTAN(1/27))−61.39xCOS(ARCTAN(1/3))+320
Y=162z−176.2ySIN(ARCTAN(1/27))−61.39xSIN(ARCTAN(1/3))+196

or

X=320−58.2x+176.1y
Y=196−19.4x−6.5y+162z

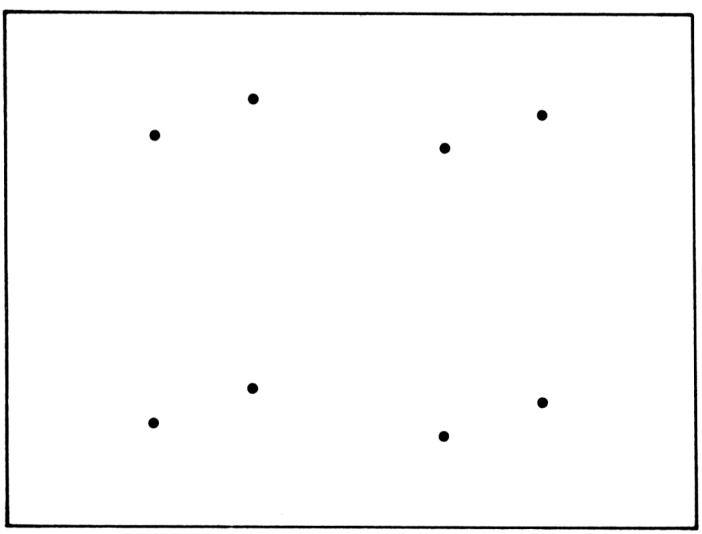
x, y and z are measured from O and X and Y from the bottom left of the screen. Ox, Oy and Oz are taken as a length of unity.

* The exact value of the aspect ratio may differ from this value if the X and Y magnifications in the TV screen monitor are set differently. It should be determined experimentally by drawing a true square symmetrically on the screen.

EXAMPLE

The following example program illustrates the use of these relationships but other examples can be found in 'CUBIC CRYSTALS' and 'UNIVERSAL ROTATION'. The program simply plots at the corners of a cube of side 2 as projected.

```
10 REM Projection
20 INPUT "Magnification (<1)";M
30 DEF FN X(X,Y,M)=320-(58.24*X-176.08*Y)*M
40 DEF FN Y(X,Y,Z,M)=196-(19.41*X+6.52*Y-162*Z)*M
50 DIM A[8,3]
60 A[1,1]=1: A[1,2]=1: A[1,3]=1
70 A[2,1]=1: A[2,2]=1: A[2,3]=-1
80 A[3,1]=1: A[3,2]=-1: A[3,3]=1
90 A[4,1]=1: A[4,2]=-1: A[4,3]=-1
100 A[5,1]=-1: A[5,2]=-1: A[5,3]=-1
110 A[6,1]=-1: A[6,2]=-1: A[6,3]=1
120 A[7,1]=-1: A[7,2]=1: A[7,3]=-1
130 A[8,1]=-1: A[8,2]=1: A[8,3]=1
140 CLS
150 FOR P=1 TO 8
160 PLOT FN X(A[P,1],A[P,2],M),FN Y(A[P,1],A[P,2],A[P,3],M)
170 NEXT
180 END
```



61. PUSH and POP

These related subroutines store values on a stack using the 'last in, first out' principal. The stack pointer SP is used to indicate which value is at the top of the stack but the actual values themselves are not transferred. The subroutine is used in the 'INORDER SEQUENCE' program to store the vertices which are occupied along the path to an empty vertex.

```
10 REM Push and Pop
20 INPUT K: DIM D[K]: REM V refers to some element of array
D
30 DIM S[K]: SP=K
40 REM Push
50 S[SP]=V: SP=SP-1: RETURN
60 REM Pop
70 V=S[SP+1]: SP=SP+1: RETURN
```

62. PYTHAGORIAN WHOLE NUMBERS

This subroutine works out the pairs of numbers less than a given value whose squares, when added together, give a perfect square e.g. 3 and 4.

To prevent covering the same numbers twice e.g. 3 and 4, and 4 and 3, the loop in B begins at A. The other point to note is the need to add a very small quantity to cover the inaccuracies in the calculations. Although a number might be printed out as a whole number on the screen, when a comparison is done the number may be one less plus .9999999999 etc. (in binary) and INT rounds down thus failing to pick up the equality in 'IF ABS(T−INT(T))=0'.

```
10 REM Pythagorian whole numbers
20 INPUT "Maximum value to be considered";N
30 FOR A=1 TO N: FOR B=A TO N
40 T=SQR(A*A+B*B)
50 IF ABS(T-INT(T+0.00001))<0.00001 THEN PRINT TAB(2);A;TAB(
10);B;TAB(20);ROUND(T,0)
60 NEXT B: NEXT A
```

63. QUADSOL

This is the straightforward solution of a quadratic equation but made idiot–proof. It works out the solution(s), both real and imaginary, of a quadratic equation of the form $ax^2+bx+c=0$.

```
10 REM Quadsol
20 NL$=CHR$(10)+CHR$(13)
30 INPUT "a=";A
40 INPUT "b=";B
50 INPUT "c=";C
60 IF A=0 AND B=0 THEN 30
70 GOSUB 1000: GOSUB 2000: END
1000 IF A=0 THEN B$="One and only": X=-C/B: RETURN
1010 P=-B/2/A: Q=(B*B-4*A*C)/4/A/A
1020 IF Q>=0 THEN B$="Real": X1=P+SQR(Q): X2=P-X1+P: RETURN
1030 B$="Imaginary": R$=STR$(P): I$=STR$(SQR(ABS(Q)))+"*i"
1040 RETURN
2000 CLS: Z$=LEFT$(B$,1): PRINT B$;" root";
2010 IF Z$="O" THEN PRINT " is ";X
2020 IF Z$="R" THEN PRINT "s are ";NL$;X1;NL$;"and";NL$;X2
2030 IF Z$="I" THEN PRINT "s are ";NL$;R$+"+"+I$;NL$;"and";N
L$;R$+"-"+I$
2040 RETURN
```

EXAMPLES

```
run
a=? 0
b=? 5
c=? 2
One and only root is -.4
Ready
run
a=? 1
b=? -5
c=? 6
Real roots are
3
2
Ready
run
a=? 1
b=? 8
c=? 20
Imaginary roots are
-4+ 2*i
-4- 2*i
Ready
```

64. REGRESSION

A group of simultaneous equations which are linearly independent can only be solved exactly if the number of equations is the same as the number of unknown variables. If there are less equations than variables then only relationships between the ratios of the variables can be found but, if there are more equations than variables then it is possible to find a unique solution which minimises the errors in the equations in terms of a least squares concept.

The subroutine given here is based on Bauer's elimination method using weighted row combinations. The A matrix is copied into the U matrix in order to preserve it and the Q matrix, which is an upper right triangular matrix, is derived by a decomposition of U using an orthogonalisation process with weighted row combinations. The answer, which is derived by back substitution, is stored in the X matrix.

Very complicated to explain in a few lines.

If M and N are equal, the routine solves the simultaneous equation in the normal way giving an exact solution.

```
10 REM Solution of N equations with M unknowns (M<=N)
20 DEFINT H,I,J,L,M,N
30 DEF FN A$(A)=MID$(STR$(A),2)
40 INPUT "Number of equations";N: IF N<=0 THEN 40
50 INPUT "Number of unknowns";M: IF M>N THEN 40
60 DIM A[N,M],B[N],X[M]
70 FOR I=1 TO N: FOR J=1 TO M
80 PRINT "A(";FN A$(I);",";FN A$(J);")=";: INPUT A[I,J]: NEX
T
90 PRINT "B(";FN A$(I);")=";: INPUT B[I]: NEXT
100 GOSUB 1000: GOSUB 2000: END
1000 DIM Q[M*(M+1)/2],U[N,M]
1010 FOR I=1 TO N: FOR J=1 TO M
1020 U[I,J]=A[I,J]: NEXT J,I
1030 L=0: FOR I=1 TO M
1040 S=0: FOR J=1 TO N: S=S+U[J,I]*U[J,I]: NEXT
1050 L=L+1: Q[L]=S
1060 T=0: FOR J=1 TO N: T=T+U[J,I]*B[J]: NEXT
1070 X[I]=T
1080 FOR H=I+1 TO M: GOSUB 1150: NEXT
1090 NEXT
1100 FOR I=M TO 1 STEP -1
1110 H=L-I: S=X[I]
1120 FOR J=I+1 TO M: S=S-Q[J+H]*X[J]: NEXT
1130 X[I]=S/Q[L]: L=L+I-M-2
1140 NEXT: RETURN
```

18.22	8.17	0	0	0	1.141	4.358
18.48	6.16	0	0	0	1.161	4.307
20.45	8.19	0	0	0	1.251	4.737
15.29	8.36	0	0	0	0.991	3.819
16.95	10.15	0	0	0	1.075	4.157
18.52	8.03	0.54	0	0	1.139	4.650
18.24	7.97	1.06	0	0	1.109	4.837
18.15	7.88	1.64	0	0	1.096	4.950
18.12	7.81	2.11	0	0	1.080	5.168
18.48	7.98	0	0.43	0	1.140	4.601
18.42	8.00	0	0.95	0	1.117	4.881
18.28	7.92	0	1.39	0	1.095	5.084
18.33	7.83	0	1.97	0	1.084	5.300
18.25	7.93	0	0	0.99	1.122	4.656
18.15	7.85	0	0	2.13	1.088	5.072
17.86	7.77	0	0	3.28	1.045	5.481
17.76	7.63	0	0	4.31	1.019	5.819
17.49	7.54	0	0	5.54	0.983	6.182

(b) As used for independent test.

%Cr	%Ni	%Nb	%Ti	%Mo	Intensity Chromium,I	Calculated Percent Cr
15.2	6.26	0	0	0	0.992	15.09
12.8	12.45	0	0	0	0.833	12.50
18.7	9.49	0	0	0	1.154	18.61
18.5	9.33	0	0	0	1.143	18.38
18.41	8.97	0	0	0	1.155	18.53
19.93	7.46	0	0	0	1.228	20.01
16.70	10.15	0	0	0	1.055	16.60
16.18	9.24	0	0	0	1.024	15.95
18.24	8.00	0	0	0	1.134	18.09
18.56	10.18	0	0	0	1.158	18.68
18.10	7.86	0	0	0	1.126	17.92
18.04	8.28	0	0	0	1.131	18.01
19.74	9.16	0	0	0	1.205	19.68
17.80	9.52	1.02	0	0	1.083	17.71
18.45	9.47	0	0.46	0	1.124	18.33
17.60	9.48	0	0	2.43	1.046	17.56

```
1150 T=0: FOR J=1 TO N: T=T+U[J,I]*U[J,H]: NEXT
1160 L=L+1: Q[L]=T: T=T/S
1170 FOR J=1 TO N: U[J,H]=U[J,H]-U[J,I]*T: NEXT
1180 RETURN
2000 FOR I=1 TO M: PRINT "X(";FN A$(I);")=";X[I]: NEXT
2010 ERASE Q,U: RETURN
```

EXAMPLE

The regression technique can be applied where a number of factors each have a linear effect on the value of some measurement. For example, the tensile strength of steel depends very strongly on the carbon content but not entirely so and the other elements present have a lesser effect which can be expressed by a formula of the type.

Tensile strength = a*Carbon% + b*Manganese% + c*Silicon% + etc.

To find the coefficients a, b, c etc. in this equation by a regression technique, you require a large number of steels of known composition and tensile strength and you solve the set of simultaneous equations to produce the best fit between the calculated and measured strength, using the above routine.

Another example which is illustrated in detail is that of doing chemical analysis using the characteristic X−rays produced when you irradiate a surface with a beam of electrons or X−rays. To a first approximation the measured signal is proportional to the concentration of the element but all the other elements present interfere and it is possible to set up an equation to give a linear relationship representing the effects.

The table shows the analysis of a group of stainless steels and the intensities of the chromium X-rays which were measured.These results are taken from an Open Report published by the British Iron and Steel Research Association (now part of British Steel Corporation).

Analysis of Stainless Steels

(a) As used for deriving the regression coefficients.

%Cr	%Ni	%Nb	%Ti	%Mo	Intensity Chromium,I	%Cr/I−11.61
4.94	0	0	0	0	0.391	1.024
9.99	0	0	0	0	0.723	2.207
14.82	0	0	0	0	0.999	3.225
19.80	0	0	0	0	1.244	4.306
24.91	0	0	0	0	1.470	5.336

The graphs 64.1 and 64.2 show the before and after relationships. Initially, the points appear to be unrelated but after correction there is a good correlation between chemical content and calculated content.

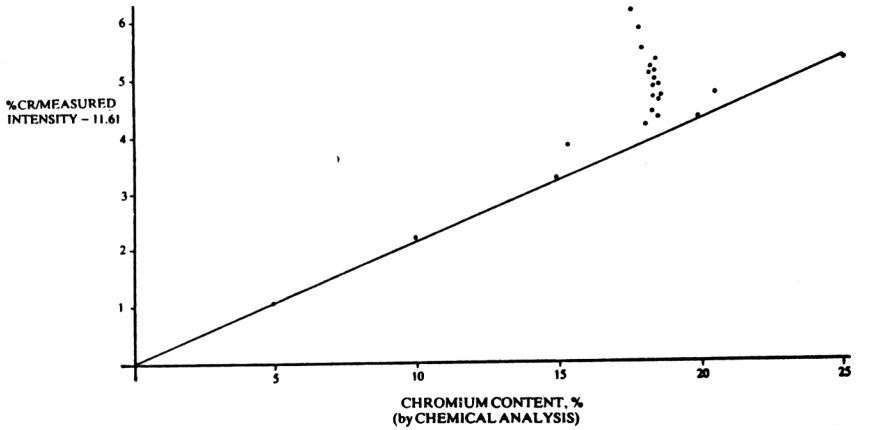


Fig. 64.1 Uncorrected Readings on Standards

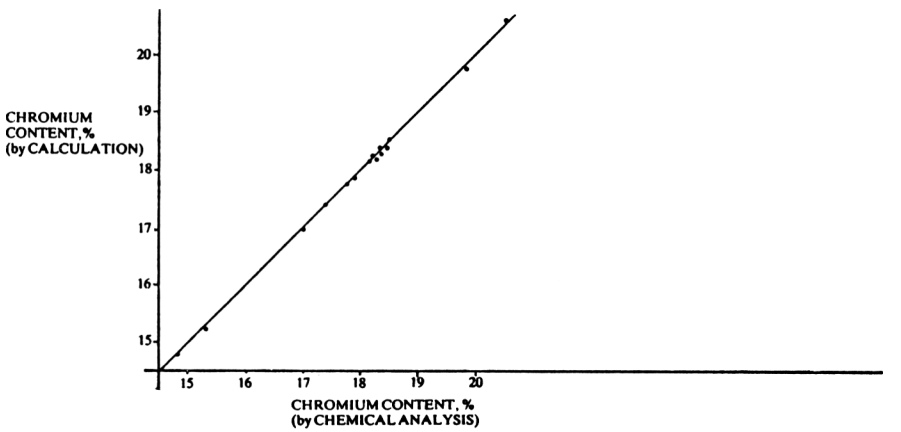


Fig. 64.2 Actual Percent Cr v.s. Calculated Percent Cr on Standards
(Data as per Fig. 64.1)

The linear relationship is of the form

$$A*\%Cr+B*\%Ni+C*\%Nb+D*\%Ti+E*\%Mo=(\%Cr/I-11.61)$$

This appears to be complicated but it arises this way because the simplest alloys you can have namely Iron plus Chromium give a curved graph to begin with. You can easily verify the correctness of the form of the equation by plotting just the first five points.

The 23 equations represented by the data were solved with the regression routine and gave the following coefficients

ELEMENT	*COEFFICIENT*
Chromium	0.21578
Nickel	0.05112
Niobium	0.40917
Titanium	0.49890
Molybdenum	0.36674

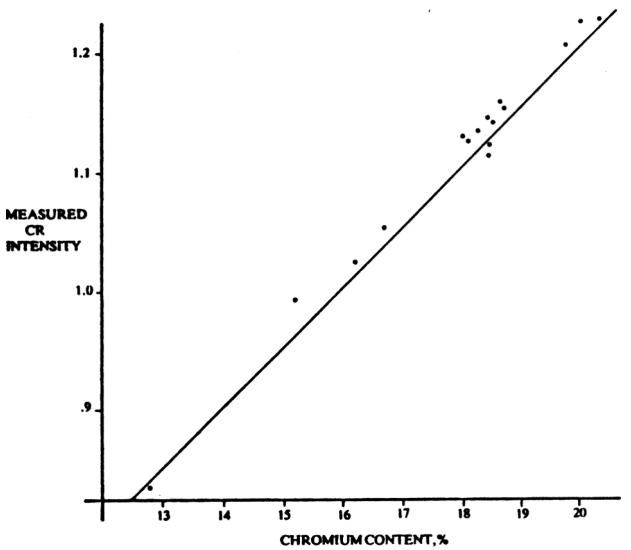


Fig.64.3 Uncorrected Readings from Steel Samples

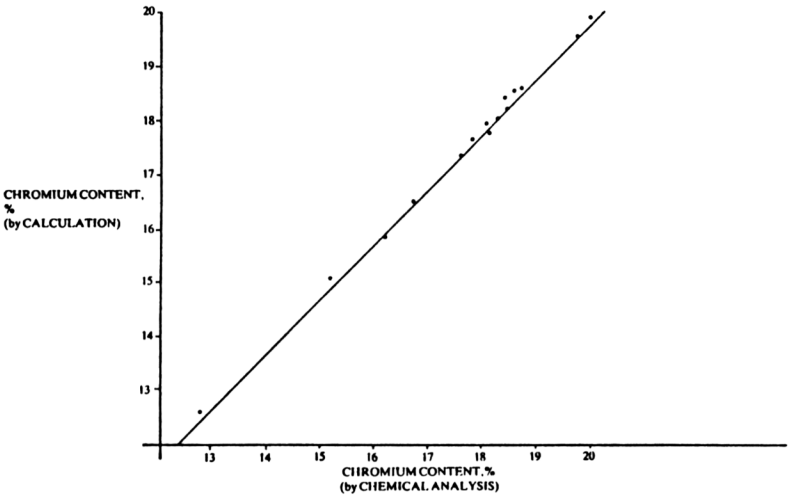


Fig. 64.4 Readings from Steel Samples After Correction

However, the true test is to take another unrelated set of steels and apply the coefficients to the measurements taken on them. This is shown in graphs 64.3 and 64.4 and it can be seen that after correction the analyses are accurate to ±0.15% Cr. Bearing in mind that there are errors in the X−ray measurements and in the chemical analyses, this result is satisfactory.

A word of warning however, to get sensible coefficients you must have a good range of variation in the factors and you should never try to calculate corrections outside the range from which the regression coefficients were derived in the first place. Always try to check your equation with a separate set of independent figures

65. ROTATION OF POINTS AROUND THE ORIGIN

This subroutine takes a group of x,y coordinates and calculates their new positions as the group is rotated about the origin into a given number of equally spaced positions. It was developed for CNC milling machines producing shapes with an axis of symmetry such as sprockets or gear teeth. The x, y coordinates are the break points where the profile changes curvature.

The angle of rotation is calculated (as theta) from the number of sectors. S[2,B] stores the list of x, y coordinates. R is the rotation matrix and I is a unit matrix used to build up the answer array T.

On the first cycle of the outer I loop, the original coordinates are stored in T. The I matrix is then changed into a θ rotation matrix and multiplies the original coordinates to produce the first shifted position. I is then transformed into a 2θ rotation matrix and so on.

Note the use of ROUND to restrict the number of decimal places PRINTed and of ERASE to enable DIM to be used in the program and hence avoid having to zero all the values in the arrays each time.

To rotate about a point (A,B) instead of the origin, subtract A from x coordinates and B from the y coordinates during the INPUT routine. After carrying out the rotation add A and B to the transformed x and y coordinates respectively. (Cf. 'UNIVERSAL ROTATION')

```
10 REM Rotation of points about the origin
20 DEFINT B,J,K,L,N: DEF FN A$(A)=MID$(STR$(A),2)
30 INPUT "No. of sectors";N: IF N<=0 THEN 30
40 INPUT "No. of points ";B: IF B<=0 THEN 40
50 DIM S[2,B]
60 THETA=2*PI/N
70 CT=COS(THETA): ST=SIN(THETA)
80 CLS: PRINT "Input the x and y coordinates"
90 FOR P=1 TO B
100 LOCATE 1,P+3: PRINT "x";FN A$(P);"=";: INPUT S[1,P]: LOC
ATE 20,P+3: PRINT "y";FN A$(P);"=";: INPUT S[2,P]
110 NEXT
120 PRINT: PRINT "          OK?  (y/n)";: INPUT K$: K$=UPPER$(
K$)
130 IF K$<>"Y" THEN ERASE S: CLS: GOTO 30
140 GOSUB 1000: GOSUB 2000: END
1000 DIM R[2,2],I[2,2],T[2,B,N]
1010 R[1,1]=CT: R[2,2]=CT: R[1,2]=-ST: R[2,1]=ST: I[1,1]=1:
I[2,2]=1
1020 FOR I=1 TO N: FOR J=1 TO B: FOR K=1 TO 2: FOR L=1 TO 2
```

```
1030 T[K,J,I]=T[K,J,I]+I[K,L]*S[L,J]
1040 NEXT L,K,J
1050 FOR J=1 TO 2: DIM Y[2]
1060 FOR K=1 TO 2: FOR L=1 TO 2
1070 Y[K]=Y[K]+R[K,L]*I[L,J]
1080 NEXT L,K
1090 FOR K=1 TO 2: I[K,J]=Y[K]
1100 NEXT K: ERASE Y: NEXT J,I
1110 RETURN
2000 CLS
2010 FOR I=1 TO N: FOR J=1 TO B
2020 PRINT "(X";FN A$(J);"),";FN A$(I);"=";ROUND(T[1,J,I],6)
2030 PRINT "(Y";FN A$(J);"),";FN A$(I);"=";ROUND(T[2,J,I],6)
2040 NEXT J,I
2050 ERASE T,I,R
2060 RETURN
```

EXAMPLE

```
No of sectors? 4
Number of points? 1
Input the X and Y coordinates
X 1 ? 10          Y 1 ? 20
OK?   Y/N

(X1),1 = 10
(Y1),1 = 20
(X1),2 = -20
(Y1),2 = 10
(X1),3 = -10
(Y1),3 = -20
(X1),4 = 20
(Y1),4 = -10
```

66–69. ROUNDING NUMBERS

The INT(X) operation always rounds down and the FIX(X) function gives the integer part of a number.A function of the type INT(X*1000+.5)/1000 will round off excess decimal places (to three in this case) but there are occasions when other rounding operations are required. ROUND(X,N) is available for rounding to N places.

66. ROUNDING UP TO AN INTEGER

```
10 REM Rounding up to an integer
20 INPUT X
30 GOSUB 1000: GOSUB 2000: END
1000 IF X=INT(X) THEN RETURN
1010 X=1+INT(X): RETURN
2000 PRINT X: RETURN
```

67. ROUNDING TO THE NEAREST INTEGER

This program is equivalent to ROUND(X,O).

```
10 REM Rounding to the nearest integer
20 INPUT X
30 GOSUB 1000: GOSUB 2000: END
1000 X=INT(X+0.5): RETURN
2000 PRINT X: RETURN
```

68. ROUNDING TO N DECIMAL PLACES

For printing purposes it is usually possible to invoke the PRINT USING statement to specify the number of decimal places. However there may be occasions when the following subroutine is better, particularly when you don't know the size of the integer part. ROUND(X,N) is almost identical but it does not supply any extra zeros.

```
10 REM Rounding to n decimal places
20 INPUT "Number";A$: INPUT "Number of decimal places";N
30 GOSUB 2000: END
2000 IF N=0 THEN PRINT INT(VAL(A$)+0.5): RETURN
2010 L=LEN(A$)
2020 FOR P=1 TO L: IF MID$(A$,P,1)="." THEN 2040
2030 NEXT: PRINT A$+"."+STRING$(N,48): RETURN
2040 IF N+P>=L THEN PRINT A$+STRING$(N+P-L,48): RETURN
2050 PRINT LEFT$(A$,P+N-1)+MID$(STR$(INT(VAL(MID$(A$,P+N))/1
0^(L-N-P)+0.5)),2)
2060 RETURN
```

69. ROUNDING TO N SIGNIFICANT FIGURES

Quoting numbers to a significant number of figures is a good way of indicating the accuracy as it can be assumed that the number is accurate to one digit in the last decimal place or non−zero digit for an integer.

The subroutine first deals with the trivial case of N=0. It then separates numbers containing a decimal point from integers. The latter either need truncating or extra zeros after a decimal point adding on. At 2050, decimal numbers greater than one which need augmenting are dealt with and then for numbers less than one the position of the decimal point is found in SUB 2080. Numbers are then truncated or augmented as necessary. STRING$(N,48) supplies the zeros.

```
10 REM Rounding to n significant figures
20 INPUT "Number";A$: INPUT "Number of significant figures";
N
30 GOSUB 2000: END
2000 IF N=0 THEN PRINT "0": RETURN
2010 L=LEN(A$)
2020 FOR P=1 TO L: IF MID$(A$,P,1)="." THEN 2050
2030 NEXT: IF N>=L THEN PRINT A$+"."+STRING$(N-L,48): RETURN
2040 PRINT LEFT$(STR$(INT(VAL(LEFT$(A$,N+1))/10+0.5)),N+1)+S
TRING$(L-N,48): RETURN
2050 IF N>=L AND VAL(A$)>1 THEN PRINT A$+STRING$(N-L+1,48):
RETURN
2060 IF VAL(A$)<1 THEN GOSUB 2080: IF N>L-Q+1 THEN PRINT A$+
STRING$(N+Q-L-1,48): RETURN ELSE PRINT INT((VAL(A$))*10^(Q+N
-P-1)+0.5)/10^(Q+N-P-1): RETURN
2070 PRINT INT((VAL(A$))*10^(N-P+1)+0.5)/10^(N-P+1): RETURN
2080 FOR Q=P+1 TO L: IF MID$(A$,Q,1)<>"0" THEN RETURN
2090 NEXT: RETURN
```

70. RUBOUT (OR FILL IN)

This subroutine rubs out everything inside a triangle or paints over the paper with INK 1 colour. The heart of the program is very simple and uses the PLOT command to over print pixels with the background colour. The complications arise from covering all orientations of the triangle including those with horizontal and vertical sides.

At 1000, the corners are relabelled so that 1 is the highest and 3 the lowest point on the screen. A loop is set up to travel from Y1 to Y3 and a horizontal box one pixel deep is plotted across the triangle in background colour. Before the loop begins it is necessary to work out the slopes of the sides without generating 'Division by zero' and then to define P12, P23, and P31 – the reciprocals of the slopes.

T determines which way the PLOT goes R to L or L to R and INT(1+ ABS(P31)) etc. is to stop the lines of the triangle itself being erased. Reversing the sign of these terms or omitting them will remove the triangle if required.

Any polygon shape is made up of triangles and the subroutine can be extended to other shapes.

```
10 REM Rubout or Fill in
20 BORDER 15: INK 0,5: INK 1,20: INK 2,24
30 A$="This is an eradicator program which rubs out the pixe
ls inside the triange by using the PLOT function to paint ov
er the paper with INK 1 colour"
40 CLS: PEN 2: PAPER 0: PRINT " Type in the coordinates of t
he corners of the triangle"
50 LOCATE 1,4: PRINT "X1=";: INPUT X1: LOCATE 20,4: PRINT "Y
1=";: INPUT Y1
60 LOCATE 1,8: PRINT "X2=";: INPUT X2: LOCATE 20,8: PRINT "Y
2=";: INPUT Y2
70 LOCATE 1,12: PRINT "X3=";: INPUT X3: LOCATE 20,12: PRINT
"Y3=";: INPUT Y3
80 CLS: GOSUB 1000: END
1000 IF Y2>Y3 THEN SW=X2: X2=X3: X3=SW: SW=Y2: Y2=Y3: Y3=SW
1010 IF Y1>Y2 THEN SW=X1: X1=X2: X2=SW: SW=Y1: Y1=Y2: Y2=SW:
 GOTO 1000
1020 FOR P=1 TO 7: PRINT A$: NEXT
1030 PLOT X1,Y1: DRAW X2,Y2: DRAW X3,Y3: DRAW X1,Y1
1040 IF X1<>X2 THEN M12=(Y1-Y2)/(X1-X2): GOTO 1060
1050 P12=0: GOTO 1070
1060 IF Y1<>Y2 THEN P12=1/M12
1070 IF X2<>X3 THEN M23=(Y2-Y3)/(X2-X3): GOTO 1090
1080 P23=0: GOTO 1100
```

```
1090 IF Y2<>Y3 THEN P23=1/M23
1100 IF X3<>X1 THEN M31=(Y3-Y1)/(X3-X1): GOTO 1120
1110 P31=0: GOTO 1130
1120 IF Y3<>Y1 THEN P31=1/M31
1130 IF Y1=Y2 THEN 1190 ELSE T=SGN(P12-P31)
1140 FOR Y=Y1+1 TO Y2+(Y2=Y3)*2
1150 X=X1-Y1*P31+Y*P31+T*INT(1+ABS(P31))
1160 Z=T*(ABS((Y1-Y)*(P31-P12))-T*INT(1+ABS(P12))+T*INT(1+AB
S(P31)))
1170 PLOT X,Y: DRAWR Z,0
1180 NEXT Y
1190 IF Y2=Y3 THEN RETURN ELSE T=SGN(P31-P23)
1200 FOR Y=Y2+1-(Y1=Y3)*2 TO Y3-1
1210 X=X1-Y1*P31+Y*P31+T*INT(1+ABS(P31))
1220 Z=T*(ABS(X2-X1+Y*(P23-P31)+Y1*P31-Y2*P23)+T*INT(1+ABS(P
31))-T*INT(1+ABS(P23)))
1230 PLOT X,Y: DRAWR Z,0
1240 NEXT: RETURN
```

```
  w i t h   I N K   1   c o l o u r
i s   a n   e r a d i c a t o r   p r o g r a m   w h i c h   r u b s
t h e   p i x e l s   i n s i d e   t h e   t r i a n g l e   b y   u
t h e   P L O T   f u n c t i       t o   p a i n t   o v e r   t h e
  w i t h   I N K   1   c o '                 - r a m   w h i c h   r u b s
i s   a n   e r a d i c a t                     t r i a n g l e   b y
t h e   p i x e l s   i n                          n t   o v e r   t h e
t h e   P L O T   f u n r
  w i t h   I N K   1
i s   a n   e r a d i '                                       c h   r u b s
t h e   p i x e l s                                             ;   b y   u
t h e   P L O T   f                                           v e r   t h e
  w i t h   t h e
i s   a n   e r a                                       m   w h i c h   r u b s
t h e   p i x e                                   t r i a n g l e   b y   u
t h e   P L O T                           . o   p a i n t   o v e r   t h e
  w i t h   I                       J r
i s   a n                       u r   p r o g r a m   w h i c h   r u b s
t h e   p i                   n s i d e   t h e   t r i a n g l e   b y   u
t h e   P               u n c t i o n   t o   p a i n t   o v e r   t h e
  w i t h       . K   1   c o l o u r
i s   a       e r a d i c a t o r   p r o g r a m   w h i c h   r u b s
t h e   p i x e l s   i n s i d e   t h e   t r i a n g l e   b y   u
t h e   P L O T   f u n c t i o n   t o   p a i n t   o v e r   t h e
  w i t h   I N K   1   c o l o u r
```

Fig. 70.1 Illustration of 'Rubout'

71. SAVING MEMORY

The Amstrad 464 has 43903 bytes available for programming and this should be enough for most purposes. However, as your skills develop in BASIC and machine code, you may be writing programs where memory is at a premium. But, by taking relatively simple steps, you can save significant amounts of memory easily.

It is sensible to record the size of major programs and, if they are getting too big, aim to reduce them. PRINT HIMEM−FRE(0) is useful for this and can be added at the end of the program.

Apart from readability, there is no point in having long names as each letter has to be interpreted. One letter with or without a digit gives 286 variables and two letters 667 (avoiding IF, FN, ON, PI etc.) which should be enough! The judicious use of X, X% and X$ as three separate variables may combine the two symbol name with the correct type.

In general however, it is better to use the DEFINT, DEFREAL and DEFSTR at the beginning of the program and choose say A−H as strings, I−T as integers and U−Z as reals. In specific cases the definition can be overridden by a statement. For example, DEFINT A: DIM A![N,M] ensures that A1, A2 etc. are integers but array A![N,M] can still hold reals.

The use of DEF FN pays off if the same expression has to be used more than once in a program though the amount of space saved depends on the complexity of the expression. A word of warning, however, in Amstrad BASIC it is not possible to use a defined function as the argument for a DIM statement or as an array variable. For example

```
DEF FN A(A)=A*A
DIM X[FN A(4)]
```

does not work and the use of a FN in an array variable upsets the FOR−−−−NEXT loops, the GOSUB−−−−RETURN s and the WHILE−−−−WENDs by altering the stack pointer. This results in UNEXPECTED NEXT, RETURN or WEND messages.

So, do not use statements such as

```
Z=X[FN A(2)]
```

but introduce another variable

```
A=FN A(2): Z=X[A]
```

instead.

The use of GOSUB saves space if the routine is used more than once but the use of ELSE with IF----THEN often avoids having to use a GOSUB at all say just to get the logic right.

Arrays use a lot of memory. For example, in 'CUBIC CRYSTAL' I(3,8,6) uses 590 in integer mode and 1250 in reals but fortunately the space can be re-used later in the program with the aid of ERASE after the data is no longer required.

You cannot store variables in DATA statements in Amstrad so the correct type must be specified in the READ instruction. See 'STRING STORE' for the most efficient methods of storing data.

Always re-use variable names if possible e.g. for non-nested loops. For example,

```
FOR A=1 TO 10: PRINT A: NEXT
FOR B=1 TO 20: PRINT B: NEXT
```

uses 9 more bytes than

```
FOR A=1 TO 10: PRINT A: NEXT
FOR A=1 TO 20: PRINT A: NEXT
```

because the last value of B is stored as well as that of A.

Multi-statement lines save memory and if not too long, do not detract from readability.

REM statements are very valuable to make a program understandable but are best placed in the printed version of the program rather than the one residing in memory. In the programs in the book REM statements are often put on lines ending in a 9 so that they can be ignored with the AUTO input facility. GOTO's and GOSUB's are to the line after the REM line and this will be a multiple of 10.

72. SCROLL

This routine should be included in any program which generates a lot of data which are PRINTed on the screen so that the rather haphazard process of stopping and starting with <ESC> can be avoided.

The routine uses VPOS(#0)to detect the current printing position and stops when it reaches 20. After clearing the screen, the printing can be restarted at will.

```
10 REM Scroll
20 INPUT "Number of values";N: REM N>25 to be relevant
30 DIM A[N]
40 FOR P=1 TO N: A[P]=P: NEXT: REM For example
50 GOSUB 2000: END
2000 CLS: FOR P=1 TO N: PRINT "P=";P,"P^2=";A[P]*A[P]: REM F
or example
2010 IF VPOS(#0)>20 THEN GOSUB 2030: CLS
2020 NEXT: RETURN
2030 LOCATE 1,22: PRINT "Press any key to continue"
2040 IF INKEY$="" THEN 2040
2050 RETURN
```

73–76. SERIES

Series can usually be written as a looping operation which is quicker to calculate than a set of individual terms. Thus the exponential series can be written as

$$S = 1 + \frac{x}{2}(1 + \frac{x}{3}(1 + \frac{x}{4}(1 + x(1 + \text{ etc}$$

the geometric series as

$$S = a(1 + r(1 + r(1 + r(1 + \text{ etc}$$

the arithmetic series as

$$S = n*a + d(1 + 2 + 3 + 4 + \text{ etc}$$ and

the binomial series as

$$S = 1 + \frac{nx}{1}(1 + \frac{(n-1)}{2}x(1 + \frac{(n-2)}{3}x(1 + \text{ etc}$$

The form of the loop used becomes obvious by inspection of the above equations.

This group of subroutines calculates the sum to N terms of several different series. (Used to study convergence and calculation of errors caused by ignoring higher order terms).

73. EXPONENTIAL SERIES

$S = 1 + x/1! + x^2/2! + X^3/3! + X^4/4! +$

```
10 REM Exponential series
20 INPUT "X=";X: INPUT "N=";N
30 GOSUB 1000: GOSUB 2000: END
1000 S=0: E=1
1010 FOR P=1 TO N
1020 S=S+E: E=E*X/P: NEXT
1030 RETURN
```

```
2000 PRINT TAB(8);"S=";S
2010 RETURN
```

74. GEOMETRIC SERIES

$S=a+ar+ar^2+ar^3+ar^4+$ $(=a(r^n-1)/(r-1))$

```
10 REM Geometric series
20 INPUT "A=";A
30 INPUT "R=";R
40 INPUT "N=";N
50 GOSUB 1000: GOSUB 2000: END
1000 S=0: G=A
1010 FOR P=1 TO N
1020 S=S+G: G=G*R: NEXT
1030 RETURN
2000 PRINT TAB(8);"S=";S
2010 RETURN
```

75. ARITHMETIC SERIES

$S=a+(a+d)+(a+2d)+(a+3d)+$

```
10 REM Arithmetic series
20 INPUT "A=";A
30 INPUT "D=";d
40 INPUT "N=";N
50 GOSUB 1000: GOSUB 2000: END
1000 S=0
1010 FOR P=1 TO N
1020 S=S+A+(P-1)*D: NEXT
1030 RETURN
2000 PRINT TAB(8);"S=";S
2010 RETURN
```

76. BINOMIAL SERIES

$S=1+nx+n(n-1)x^2/2!+n(n-1)(n-2)x^3/3!+$

$=(1+x)^n$ (if $|x|<1$ when n<>positive integer)

```
10 REM Binomial series
20 INPUT "X=";X
30 INPUT "N=";N
40 INPUT "Number of terms";R
50 GOSUB 1000: GOSUB 2000: END
1000 S=0: Z=1: M=N
1010 FOR P=1 TO R
1020 S=S+Z: Z=Z*M*X/P: M=M-1: NEXT
1030 RETURN
2000 PRINT TAB(8)"S=";S
2010 RETURN
```

77. SIDEPRINT

This subroutine enables numbers to be printed so that the righthand overflow is printed below the number and not on the opposite side of the screen.

Two incrementing operations are employed *viz* FOR P=1 TO LEN(N$) and C=C+1. At the edge of the screen C reverts to C=W+P−LEN(N$) to print the remainder of the number on the line below.

```
10 REM SidePrint
20 INPUT "MODE";M: IF M<0 OR M>2 THEN 20
30 MODE M: W=21-20*(M=1 OR M=2)-40*(M=2)
40 INPUT "Number";N
50 INPUT "Row";R: REM 1<=R<=23
60 INPUT "Column";C: REM 1<=C<=W-1
70 GOSUB 2000: END
2000 N$=STR$(N): L=LEN(N$)
2010 CLS: FOR P=1 TO L
2020 LOCATE C,R: PRINT MID$(N$,P,1)
2030 C=C+1
2040 IF C=W THEN R=R+1: C=W+P-L
2050 NEXT
2060 LOCATE 1,1
2070 RETURN
```

78. SIMULTANEOUS EQUATIONS

This solution uses the inverse matrix method whereby if equations, say

$$a_1*X+b_1*Y+c_1*Z=K_1$$
$$a_2*X+b_2*Y+c_2*Z=K_2$$
$$a_3*X+b_3*Y+c_3*Z=K_3$$

are written in matrix notation as

$$\begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} K_1 \\ K_2 \\ K_3 \end{bmatrix}$$

Then

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix}^{-1} * \begin{bmatrix} K_1 \\ K_2 \\ K_3 \end{bmatrix}$$

The inverse is derived by the Gauss–Jordan method and multiplied by the constants' matrix.

SUB 1060 checks for zeros along the diagonal and adds two lines together in SUB 1080 to avoid a crash in line 1110. The next line looks for a zero in the top left hand 2x2 determinant and adds the last equation to the first if it finds this. SUB 1110 is the matrix inversion routine which also checks, in line 1190, that the determinant is not zero (a precondition of being able to find the inverse). On return from SUB 1110, the inverse is multiplied by the constants' matrix E[K,N+1] and S[J] is PRINTed using CHR$(90–N+J) to print the appropriate variable names x,y,z etc..

The method is not limited to 8 x 8 equations, this limitation arises only from the Data Input routine and can easily be circumvented.

This subroutine tests whether there is a solution and if so, solves the equations.

```
10 REM Simultaneous equations
20 DEFINT I,J,K,N: DEF FN A$(A)=MID$(STR$(A),2)
30 INPUT "Number of variables";N: IF N<=1 THEN 20
40 CLS: PRINT "Insert the coefficients"
50 FOR J=1 TO N: FOR K=1 TO N+1
60 PRINT "E("+FN A$(J)+",";FN A$(K)+")=";: INPUT E(J,K)
70 NEXT K,J
80 REM Alternatively, use DATA INPUT (Linear Equations)
90 GOSUB 1000: GOSUB 2000: END
1000 DIM S[N]: GOSUB 1060: IF D=0 THEN RETURN
1010 IF N>1 THEN IF E[1,1]*E[2,2]=E[1,2]*E[2,1] THEN GOSUB 1
100
1020 GOSUB 1110: IF D=0 THEN RETURN
1030 FOR J=1 TO N: FOR K=1 TO N
1040 S[J]=S[J]+E[J,K]*E[K,N+1]
1050 NEXT K,J: RETURN
1060 D=1: FOR I=1 TO N: IF E[I,I]=0 THEN GOSUB 1080
1070 NEXT: RETURN
1080 FOR J=1 TO N: IF E[J,I]<>0 THEN FOR K=1 TO N+1: E[I,K]=
E[I,K]+E[J,K]: NEXT: RETURN
1090 NEXT: D=0: RETURN
1100 FOR I=1 TO N+1: E[1,I]=E[1,I]+E[N,I]: NEXT: RETURN
1110 FOR I=1 TO N: E[I,I]=1/E[I,I]
1120 FOR J=1 TO N: IF J=I THEN 1190
1130 E[J,I]=E[J,I]*E[I,I]
1140 FOR K=1 TO N: IF K=I THEN 1180
1150 E[J,K]=E[J,K]-E[J,I]*E[I,K]
1160 IF J<>N+(I=N) THEN 1180
1170 E[I,K]=-E[I,I]*E[I,K]
1180 NEXT
1190 NEXT: IF I<>N THEN IF ABS(E[I+1,I+1])<0.0000001 THEN D=
0: RETURN
1200 NEXT: RETURN
2000 CLS: IF D=0 THEN PRINT "There is no solution": ERASE S:
 RETURN
2010 LOCATE 7,1: PRINT "The solution is": PRINT
2020 FOR J=1 TO N: PRINT TAB(3);CHR$(90-N+J);"=";S[J]: PRINT
2030 NEXT
2040 ERASE S: RETURN
```

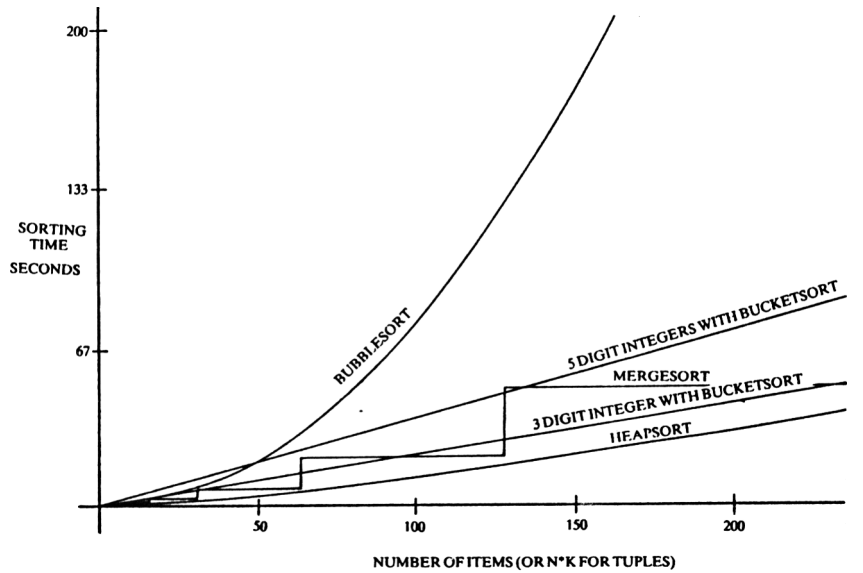
EXAMPLE

```
N=6
        Your Equations are

U/Y     V/Z     W       X           Const.

a1*u + b1*v + c1*w + d1*x +
e1*y + f1*z                         =k1
a2*u + b2*v + c2*w + d2*x +
e2*y + f2*z                         =k2
a3*u + b3*v + c3*w + d3*x +
e3*y + f3*z                         =k3
a4*u + b4*v + c4*w + d4*x +
e4*y + f4*z                         =k4
a5*u + b5*v + c5*w + d5*x +
e5*y + f5*z                         =k5
a6*u + b6*v + c6*w + d6*x +
e6*y + f6*z                         =k6


Now insert the values a1=?

U/Y     V/Z     W       X           Const.

7   +  5   +  -4   +  2 +
13  +  6                           = 15
4.5 +  2   +  19   +  24 +
-7  +  9.6                         = 3
5.5 +  13  +  9.43+  -5.46+
5   +  32                          = 5.44
-7.9+  6.1 +  3.2  +  2.7 +
5   +  8                           = 23
6.9 +  5.3 +  6.7  +  2 +
0   +  5.7                         = 4.8
-5.4+  1   +  6.8  +  4 +
3.88+  4                           = 87

The solution is

u=-1.43955734
v=-7.19042574
w= 10.7642367
x=-3.94752449
y= 9.53563634
z=-1.99704118

(Calculation time 5 seconds)
```

79–84. SORTING

Sorting words or numbers is a common operation in computing and many algorithms are available to do it. However, choosing the best one requires some thought and examples of four different algorithms are given to illustrate different choices. The main consideration is the speed of the algorithm and hence how the time increases with the number of items to be sorted.

Bubblesort is commonly used in programs but in fact is very poor for large numbers of items as the time increases as the square of the number. Heapsort is much more efficient for large numbers as the time only increases as n log n. The reasons for this can be seen by comparing the operation of the sorting algorithms. If we have a list of items which is in order except for the largest item which is at the wrong end, then with Bubblesort it moves to the correct position one move at a time, whereas with Heapsort and Mergesort it moves one place first time, then two places, four places, eight places, sixteen places etc. and gets to the correct position much faster.



Mergesort is almost as fast as Heapsort but as it requires 2^n values in the algorithm, extra values have to be added and it takes as long to sort 65 values as it does 128. The first sorting can be done during the INPUT however.

Bucketsort has long been used in punched card sorting machines and is useful for sorting integers or strings. A list of n strings or integers containing k characters can be sorted in a time proportional to k times n.

The graph shows how the four routines perform. Clearly, Bubblesort although by far the simplest routine is unsuitable for sorting large numbers of items.

Two other sorting routines are included one for words and one for angles which involve other considerations than just sorting i.e. organising the data so that they are suitable for a standard sorting algorithm.

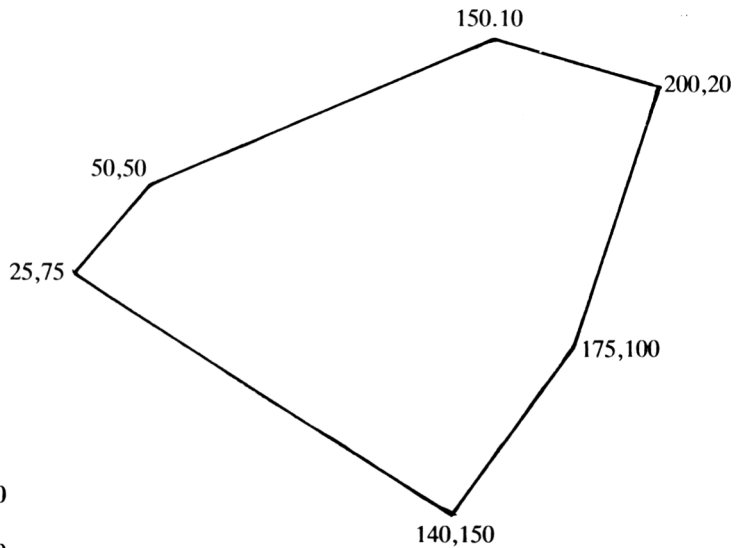
In both cases Bubblesort has been used in the program but if larger numbers of items were involved then a more efficient algorithm should be considered.

79. ANGLESORT

This subroutine puts a list of coordinates into angular order relative to their centre.

The first eight lines find the highest and lowest values of x and y and the point (XM,YM) is chosen to be halfway in between. The angles between the horizontal and the lines joining each point to this mid point are calculated and inspected to see which quadrant they are in. This is because a tangent is positive in the first and third quadrants and negative in the other two. The angle is adjusted if necessary by adding PI or 2*PI and then stored in B[N]. A bubblesort routine, 1140, is used to put the angles in order and the coordinates follow in sympathy. In the display, the 0.92 is to correct the aspect ratio of the screen.

EXAMPLE




```
150,10
50,50
200,20
175,100
25,75
140,150
```

Fig. 79.1 Anglesort

```
10 REM Anglesort
20 DEF FN A$(A)=MID$(STR$(A),2)
30 INPUT "Number of pairs of readings";N: IF N<=0 THEN 30
40 DIM A[1,N]: CLS
50 FOR P=1 TO N: LOCATE 5,P+2: PRINT "x";FN A$(P);"=";: INPU
T A[0,P]
60 LOCATE 21,2+P: PRINT "y";FN A$(P);"=";: INPUT A[1,P]: NEX
T
70 GOSUB 1000: GOSUB 2000: END
1000 XX=0: YX=0: XN=A[0,1]: YN=A[1,1]
1010 FOR P=1 TO N
1020 IF A[0,P]>=XX THEN XX=A[0,P]
1030 IF A[1,P]>=YX THEN YX=A[1,P]
1040 IF A[0,P]<XN THEN XN=A[0,P]
1050 IF A[1,P]<YN THEN YN=A[1,P]
1060 NEXT
1070 XM=(XX+XY)/2: YM=(YX+YN)/2
1080 DIM B[N]: FOR P=1 TO N
1090 Z=ATN(A[1,P]-YM)/(A[0,P]-XM)
1100 IF A[0,P]>=XM AND A[1,P]>=YM THEN B[P]=Z
1110 IF A[0,P]<XM AND A[1,P]>=YM OR A[0,P]<XM AND A[1,P]<YM
THEN B[P]=Z+PI
1120 IF A[0,P]>=XM AND A[1,P]<YM THEN B[P]=Z+2*PI
1130 NEXT
1140 Q=0: FOR P=1 TO N-1
1150 IF B[P+1]<B[P] THEN SW=B[P]: B[P]=B[P+1]: B[P+1]=SW: SW
=A[0,P]: A[0,P]=A[0,P+1]: A[0,P+1]=SW: SW=A[1,P]: A[1,P]=A[1
,P+1]: A[1,P+1]=SW: Q=Q+1
1160 NEXT: IF Q<>0 THEN 1140
1170 RETURN
2000 CLS: TAG
2010 FOR P=0 TO N-1
2020 Z1=1+P: Z2=1+(P+1)MOD N
2030 PLOT A[0,Z1],0.92*A[1,Z1]
2040 DRAW A[0,Z2],0.92*A[1,Z2]: MOVER 5,-5: PRINT FN A$(A[0,
Z2]);",";FN A$(A[1,Z2]);
2050 NEXT
2060 ERASE B: TAGOFF:
2070 RETURN
```

80. BUBBLESORT

This well known sorting routine is called bubblesort because high numbers work their way up the list like bubbles in a liquid. Q counts the number of exchanges made. Line 1010 decides whether an interchange between adjacent numbers is required and if so does it. The routine keeps on recycling until no further interchanges are needed i.e. when Q=0.

This subroutine puts numbers into descending order. To change to an ascending order replace > by < in line 1010.

```
10 REM Bubblesort
20 N=0
30 WHILE N<=0: INPUT "Number of items";N: WEND
40 DIM A[N]
50 FOR P=1 TO N: INPUT A[P]: NEXT
60 GOSUB 1000: GOSUB 2000: END
1000 Q=0: FOR P=1 TO N-1
1010 IF A[P+1]>A[P] THEN SW=A[P]: A[P]=A[P+1]: A[P+1]=SW: Q=
Q+1
1020 NEXT: IF Q<>0 THEN 1000
1030 RETURN
2000 FOR P=1 TO N: PRINT A[P]: NEXT
2010 RETURN
```

81. BUCKETSORT

In Bucketsort, the tuples are sorted first in terms of the last character. Then, after reconstituting the string, in terms of the next–to–last character and so on. For example, the following six tuples BACD, BBCD, ABCD, BBAC, BAAB, AABC when sorted give

A
B BAAB
C BBAC,AABC
D BACD,BBCD,ABCD

which gives a new order of BAAB, BBAC, AABC, BACD, BBCD, ABCD. When sorted in order of the next–to–last letter this gives

A BAAB,BBAC
B AABC
C BACD,BBCD,ABCD
D

This becomes BAAB, BBAC, AABC, BACD, BBCD, ABCD.

The next two sorts give

A BAAB,AABC,BACD
B BBAC,BBCD,ABCD
C
D

and

A AABC,ABCD
B BAAB,BACD,BBAC,BBCD
C
D

Thus the final order is AABC, ABCD, BAAB, BACD, BBAC, BBCD

The P loop works backwards through the characters. A$="" etc. empties the buckets. The Q loop finds the comma separating the tuples (See 'STRING STORAGE') and the IF statements place pointers to the appropriate tuple and these are joined together in Z$ ready for the next

cycle of the P loop.

The alternative routine uses the same input and output but is geared to sorting integers.

To Bucketsort with the whole of the alphabet it would be sensible to use a 2–3 Tree for rapid sorting. A tree of height 3 has 27 leaves which will accommodate the 26 letters of the alphabet and a space.

These can be identified with an average of five questions.

```
10 REM Bucketsort (Equal length tuples made up from A, B, C
and D)
20 DEFINT A,B,K,N,P,Q
30 INPUT "Number of tuples";N: IF N<=0 THEN 30
40 INPUT "Number of characters in each tuple";K: IF K<=0 THE
N 40
50 DIM D$[N,K]: Z$=""
60 FOR P=1 TO N: Z$=Z$+STR$(P)+","
70 PRINT "Tuple No.";P;"is";: INPUT T$
80 FOR Q=1 TO K: D$[P,Q]=MID$(T$,Q,1): NEXT Q,P
90 GOSUB 1000: GOSUB 2000: END
1000 FOR P=K TO 1 STEP -1: A=1: B=1: A$="": B$="": C$="": D$
=""
1010 FOR Q=1 TO LEN(Z$)
1020 IF MID$(Z$,Q,1)="," THEN A=Q: Z=VAL(MID$(Z$,B+(B=2),A-B
-(B=2))) ELSE 1070
1030 IF D$(Z,P)="A" THEN A$=A$+STR$(Z)+",": GOTO 1070
1040 IF D$(Z,P)="B" THEN B$=B$+STR$(Z)+",": GOTO 1070
1050 IF D$(Z,P)="C" THEN C$=C$+STR$(Z)+",": GOTO 1070
1060 IF D$(Z,P)="D" THEN D$=D$+STR$(Z)+","
1070 B=A+1: NEXT
1080 Z$=A$+B$+C$+D$
1090 NEXT
1100 RETURN
2000 A=1: B=1: FOR P=1 TO LEN(Z$)
2010 IF MID$(Z$,P,1)="," THEN A=P: Z=VAL(MID$(Z$,B+(B=2),A-B
-(B=2))) ELSE 2030
2020 FOR Q=1 TO K: PRINT D$(Z,Q);: NEXT: PRINT
2030 B=A+1: NEXT
2040 RETURN



999 REM For numbers use the following subroutine
1000 FOR P=K TO 1 STEP -1: A=1: B=1: A$="": B$="": C$="": D$
="": E$="": F$="": G$="": H$="": I$="": J$=""
1010 FOR Q=1 TO LEN(Z$)
1020 IF MID$(Z$,Q,1)="," THEN A=Q: Z=VAL(MID$(Z$,B+(B=2),A-B
-(B=2))): T$=D$(Z,P) ELSE 1130
```

```
1030 IF T$="0" THEN A$=A$+STR$(Z)+",": GOTO 1130
1040 IF T$="1" THEN B$=B$+STR$(Z)+",": GOTO 1130
1050 IF T$="2" THEN C$=C$+STR$(Z)+",": GOTO 1130
1060 IF T$="3" THEN D$=D$+STR$(Z)+",": GOTO 1130
1070 IF T$="4" THEN E$=E$+STR$(Z)+",": GOTO 1130
1080 IF T$="5" THEN F$=F$+STR$(Z)+",": GOTO 1130
1090 IF T$="6" THEN G$=G$+STR$(Z)+",": GOTO 1130
1100 IF T$="7" THEN H$=H$+STR$(Z)+",": GOTO 1130
1110 IF T$="8" THEN I$=I$+STR$(Z)+",": GOTO 1130
1120 IF T$="9" THEN J$=J$+STR$(Z)+","
1130 B=A+1: NEXT
1140 Z$=A$+B$+C$+D$+E$+F$+G$+H$+I$+J$: NEXT
1150 RETURN
```

```
10 REM Bucketsort for the alphabet using a 2-3 tree
20 DEFINT A,B,H,I,K,L,N,P,Q,V,Z
30 DIM L$[40],M$[13],T$[27]
40 L$="HBKT CFILORUX": M$="QENWADGJMPSVY"
50 FOR P=1 TO 13: L$[P]=MID$(L$,P,1): M$[P]=MID$(M$,P,1): NE
XT
60 L$[14]=" ": FOR P=15 TO 40: L$[P]=CHR$(51+P): NEXT
70 INPUT "Number of tuples";N
80 INPUT "Maximum number of characters per tuple";K
90 DIM D$[N,K]: Z$=""
100 FOR P=1 TO N: Z$=Z$+STR$(P)+","
110 PRINT "Tuple No.";P;"is";: INPUT T$
120 L=LEN(T$): IF L>K THEN 110
130 T$=UPPER$(T$)+STRING$(K-L,32)
140 FOR Q=1 TO K: D$(P,Q)=MID$(T$,Q,1): NEXT Q,P
150 GOSUB 1000: GOSUB 2000: END
1000 FOR P=K TO 1 STEP -1: A=1: B=1
1010 FOR I=1 TO 27: T$[I]="": NEXT
1020 FOR Q=1 TO LEN(Z$)
1030 IF MID$(Z$,Q,1)="," THEN A=Q: Z=VAL(MID$(Z$,B+(B=2),A-B
-(B=2))) ELSE 1070
1040 V=1: H=1
1050 WHILE H<4: GOSUB 1110: H=H+1: WEND
1060 V=V-13: T$[V]=T$[V]+STR$(Z)+","
1070 B=A+1: NEXT: Z$=""
1080 FOR I=1 TO 27: IF T$[I]<>"" THEN Z$=Z$+T$[I]
1090 NEXT I,P
1100 RETURN
1110 IF D$[Z,P]<=L$[V] THEN V=3*V-1: RETURN
1120 IF D$[Z,P]<=M$[V] THEN V=3*V: RETURN
1130 V=3*V+1: RETURN
2000 A=1: B=1: FOR P=1 TO LEN(Z$)
2010 IF MID$(Z$,P,1)="," THEN A=P: Z=VAL(MID$(Z$,B+(B=2),A-B
-(B=2))) ELSE 2030
2020 FOR Q=1 TO K: PRINT D$[Z,Q];: NEXT: PRINT
2030 B=A+1: NEXT
2040 RETURN
```

82. HEAPSORT

A 'heap' is the name for a binary tree structure where the elements are so arranged that the one associated with any vertex is greater than or equal to those associated with its two sons. The level of any leaf can only differ by one level at most from any other leaf.

The diagram shows a typical heap. If A1, A2, A3 etc. are the elements to be sorted, then they are stored in an array A[N]. The heap property implies that

$$A[I] >= A[2*I] \quad \text{for } 1 <= I <= N/2$$

and

$$A[I] >= A[2*I+1] \quad \text{for } 1 <= I < N/2$$

The first part of the routine is 'heapify' which establishes the order in the array to give the heap property.

The second part then sorts the array elements into order by
1. removing the largest element (which is in the root) and exchanging it for the last element. This element then takes no further part.

2. reforming the remaining elements using 'heapify'

3. repeating until the array contains all the elements in ascending order.

This is illustrated in the sequence of arrays. The flowcharts show how the subroutine works.

```
10 REM Heapsort
20 INPUT "Number of values";M: IF M<=1 THEN 20
30 DIM D[M]
40 FOR P=1 TO M: PRINT "Value";P;"is";: INPUT D[P]: NEXT
50 GOSUB 1000: GOSUB 2000: END
1000 FOR P=INT(M/2) TO 1 STEP -1: R=P
1010 S=R+R: T=S+1
1020 IF D[R]<D[S] THEN GOSUB 1170: GOTO 1050
1030 IF T<=M THEN IF D[R]<D[T] THEN GOSUB 1200: GOTO 1050
1040 GOTO 1060
1050 IF R<=M/2 THEN 1010
1060 NEXT P
1070 FOR P=M TO 3 STEP -1: SW=D[1]: D[1]=D[P]: D[P]=SW: R=1
1080 S=R+R: T=S+1
1090 IF D[R]<D[S] THEN GOSUB 1150: GOTO 1120
1100 IF T<P THEN IF D[R]<D[T] THEN GOSUB 1200: GOTO 1120
1110 GOTO 1130
1120 IF R<=(P-1)/2 THEN 1080
1130 NEXT P
```

```
1140 SW=D[1]: D[1]=D[2]: D[2]=SW: RETURN
1150 IF T>=P THEN 1190
1160 GOTO 1180
1170 IF T>M THEN 1190
1180 IF D[S]<D[T] THEN 1200
1190 SW=D[R]: D[R]=D[S]: D[S]=SW: R=R+R: RETURN
1200 SW=D[R]: D[R]=D[T]: D[T]=SW: R=R+R+1: RETURN
2000 FOR P=1 TO M: PRINT D[P]: NEXT
2010 RETURN
```

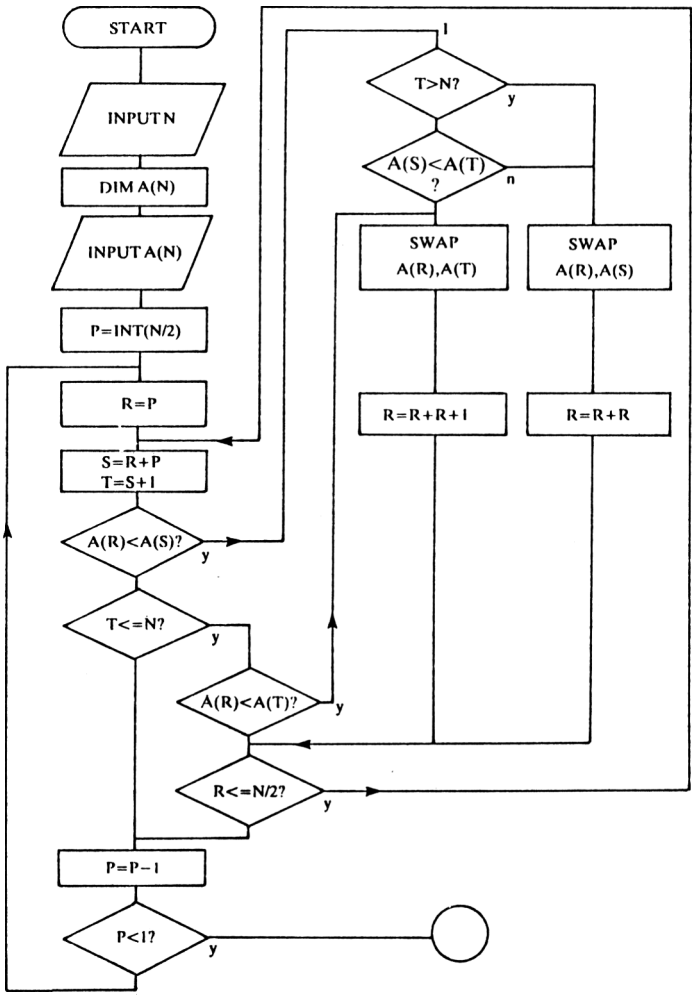


Fig. 82.1 Heapify

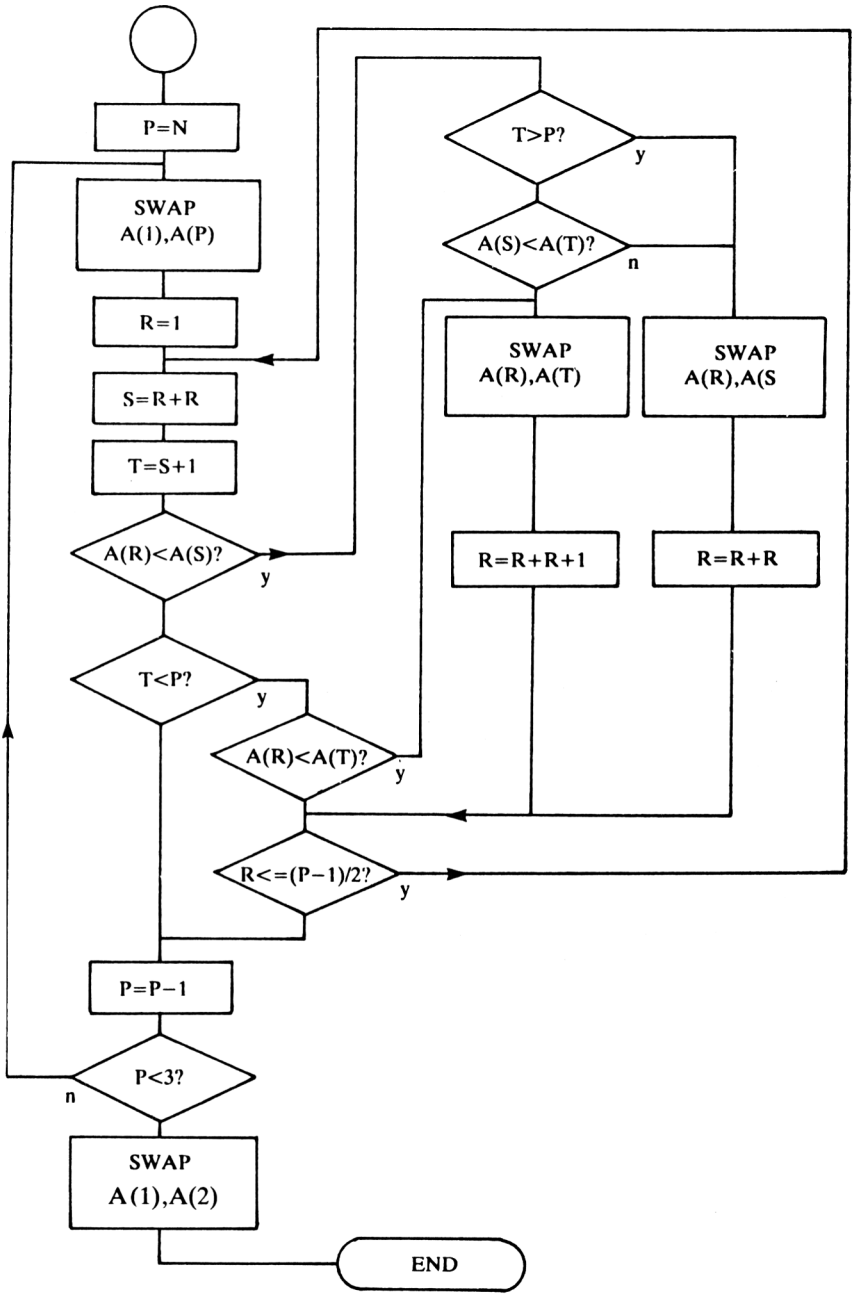

P=N
SWAP
A(1),A(P)
R=1
S=R+R
T=S+1
A(R)<A(S)?
y
T<P?
y
A(R)<A(T)?
y
R<=(P-1)/2?
y
P=P-1
P<3?
n
SWAP
A(1),A(2)
END
T>P?
y
A(S)<A(T)?
n
SWAP
A(R),A(T)
SWAP
A(R),A(S
R=R+R+1
R=R+R

Fig. 82.2 Sort

83. MERGESORT

MERGESORT is a subroutine for sorting a list of 2^n items. If the number of items is not a power of two, then values higher than the maximum in the list have to be added and discarded at the end. The routine works by first ordering pairs of values, then merging adjacent pairs, then adjacent 4's, 8's, 16's etc., using SUB Merge. For example, the list

	8,6,4,1,5,6,3,1
becomes	6 8,1 4,5 6,1 3 which
becomes	1 4 6 8,1 3 5 6 which
becomes	1 1 3 4 5 6 6 8

The DEF FN is to pack the format in the INPUT. During the INPUT, the maximum value MX is sought and pairs of values are ordered (in line 1050). SUB 1200 fills out the rest of the array A with values of MX+1. Line 1080 begins with pairs of values and merges them with SUB 1090, then it merges groups of 4, 8, 16 etc. As the extra MX+1 values finish up at the beginning of the array, they are ignored by the PRINT statement which stops at 1+S−N. To reverse the order, loop from 1+S−N TO S.

```
10 REM Mergesort
20 DEFINT A-C,N,P,X-Z: DEF FN A$(A)=MID$(STR$(A),2)
30 INPUT "Number of values";N: IF N<=0 THEN 30
40 M=N: P=0: MX=0
50 WHILE M>1: M=M/2: P=P+1: WEND
60 S=2^P: DIM A[S],X[S/2+1],Y[S/2+1],Z[S+2]
70 GOSUB 1000: GOSUB 2000: END
1000 FOR A=1 TO N: PRINT "A("+FN A$(A)+")=";: INPUT A[A]
1010 IF A[A]>MX THEN MX=A[A]
1020 A=A+1: IF A>N THEN GOSUB 1200: GOTO 1070
1030 PRINT "A("+FN A$(A)+")=";: INPUT A[A]
1040 IF A[A]>MX THEN MX=A[A]
1050 IF A[A]>A[A-1] THEN SW=A[A]: A[A]=A[A-1]: A[A-1]=SW
1060 NEXT A: GOSUB 1200: GOTO 1080
1070 IF A[N+1]>A[N] THEN SW=A[N]: A[N]=A[N+1]: A[N+1]=SW
1080 M=2: X[1]=MX+2: Y[1]=X[1]
1090 FOR B=0 TO S/M/2-1: T=2*B*M-1: FOR C=2 TO M+1
1100 X[C]=A[C+T]: Y[C]=A[C+T+M]: NEXT: GOSUB 1220
1120 NEXT: M=M+M: IF M<S GOTO 1090 ELSE RETURN
1200 IF M<>1 THEN FOR A=N+1 TO S: A[A]=MX+1: NEXT
1210 RETURN
1220 X=M+1: Y=X
1230 IF X[X]<Y[Y] THEN Z[X+Y]=X[X]: IF X>1 THEN X=X-1: GOTO
1230
1240 Z[X+Y]=Y[Y]: IF Y>1 THEN Y=Y-1: GOTO 1230
1250 FOR J=3 TO M+M+2: A[J+T-1]=Z[J]: NEXT: RETURN
2000 FOR A=S TO 1+S-N STEP -1
2010 PRINT A[A]: NEXT
2020 ERASE A,X,Y,Z: RETURN
```

84. WORDSORT

Using a sorting routine on ASCII characters puts them in order of their code. To get a proper alphabetical order it is necessary to reduce all capitals to lower case letters or vice versa. Other symbols are not relevant to alphabetical ordering and need to be ignored.

A$ is used to store the original words whilst B$ holds the reduced capitals and lower case letters but excluding non-letters. B$ is ordered by a bubblesort-line 1100 and A$ follows in sympathy. Each word is checked for the right characteristics before it is accepted for sorting.

```
10 REM Wordsort
20 DEFINT N,P,Q,T
30 INPUT "Number of words";N: IF N<=0 THEN 30
40 GOSUB 1000: GOSUB 2000: END
1000 N=N-1: DIM A$[N],B$[N]
1010 FOR Q=0 TO N: PRINT "Word";Q+1;" is ";: INPUT Z$: L=LEN
(Z$)
1020 A$[Q]=Z$: B$[Q]=Z$
1030 FOR P=1 TO L: T=1: X$=MID$(Z$,P,1)
1040 IF X$="-" OR X$="'" OR X$>="A" AND X$<="Z" OR X$>="a" A
ND X$<="z" THEN NEXT ELSE PRINT Z$;" is not a word": Q=Q-1:
GOTO 1090
1050 FOR P=1 TO L: X$=MID$(Z$,P,1)
1060 IF X$>="A" AND X$<="Z" THEN MID$(B$[Q],T,1)=LOWER$(X$):
 T=T+1 ELSE IF X$>="a" AND X$<="z" THEN MID$(B$[Q],T,1)=X$:
T=T+1
1070 NEXT
1080 B$[Q]=LEFT$(B$[Q],T-1)
1090 NEXT
1100 Q=0: FOR P=0 TO N-1
1110 IF B$[P+1]<B$[P] THEN SW$=B$[P]: B$[P]=B$[P+1]: B$[P+1]
=SW$: SW$=A$[P]: A$[P]=A$[P+1]: A$[P+1]=SW$: Q=Q+1
1120 NEXT: IF Q<>0 THEN 1100 ELSE RETURN
2000 FOR P=0 TO N: PRINT A$[P]: NEXT
2010 RETURN
```

```
run                          (output)
Number of words? 9           brown
Word   1   is  ? The         dog
Word   2   is  ? quick       fox
Word   3   is  ? fox         jumps
Word   4   is  ? jumps       lazy
Word   5   is  ? over        over
Word   6   is  ? the         quick
Word   7   is  ? lazy        The
Word   8   is  ? brown       the
Word   9   is  ? dog
```

85. STATISTICAL ANALYSIS

This subroutine calculates the statistical properties of a group of readings (Mean, standard deviation, minimum, maximum and the number of readings in each histogram interval) and displays the results as a 3–D histogram.

The number of intervals in the histogram is chosen to be near the squareroot of the number of readings to give a satisfactory display.

The first eight lines calculate the minimum, maximum, mean and standard deviation for the data. These are stored in Z$ to control the number of digits printed out on the display (by slicing). D[2,I+3] is used to store (a) the lower value of the histogram intervals in D[1,I+3] and (b) the number of values in each interval in D[2,I+3]. The highest number of values is made equal to D2MAX.

A[2,I+4] holds the values to be plotted on the histogram which is scaled to fit a line of slope 1/3 which passes through (112,34) and (352,107). These two points are made equal to the minimum and maximum values. The height is scaled to 100 by line 2030.

The following lines draw the histogram and shade it. After line 2090 the main axes are drawn and the statistical information printed on the screen.

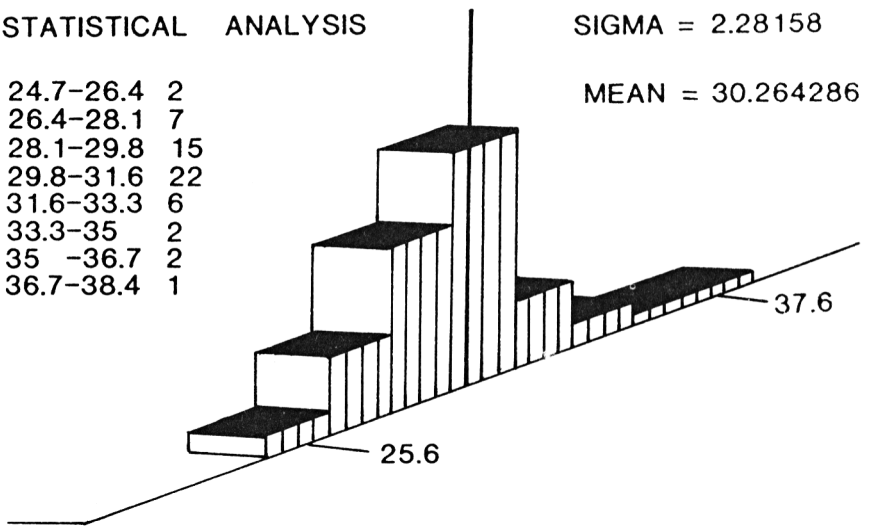
```
10 REM Statistical analysis
20 BORDER 5: INK 0,1: INK 1,15
30 DEFINT I,P,Q: DEF FN A$(A)=MID$(STR$(A),2)
40 INPUT "How many readings (min 4)";N: IF N<4 THEN 40
50 CLS: DIM V[N]
60 PRINT: PRINT "Type in the values of the readings"
70 PRINT: FOR P=1 TO N: PRINT TAB(5);"V";FN A$(P);"=";: INPU
T V[P]: NEXT
80 REM Alternatively, use 'DATA INPUT (Single Variable)' to
establish N and V(N)
90 CLS: GOSUB 1000: GOSUB 2000: END
1000 I=INT(SQR(N)): MEAN=0: S=0: XMAX=V[1]: XMIN=V[1]
1010 FOR P=1 TO N
1020 MEAN=MEAN+V[P]: IF V[P]>XMAX THEN XMAX=V[P]
1030 NEXT: MEAN=MEAN/N
1040 FOR P=1 TO N: D=V[P]-MEAN
1050 S=S+D*D: IF V[P]<XMIN THEN XMIN=V[P]
1060 NEXT: S=SQR(S/(N-1))
1070 T=(XMAX-XMIN)/I: XR=T*I
1080 DIM Z$[I+9],D[2,I+3]
1090 FOR P=6 TO I+7: Z$(P)=STR$(XMIN+(P-6.5)*T): NEXT
```

```
1100 Z$(1)=STR$(MEAN)
1110 Z$(2)=STR$(XMIN)
1120 Z$(3)=STR$(XMAX)
1130 Z$(4)=STR$(S)
1140 Z$(5)=""
1150 D2MAX=0: D[2,1]=0
1160 FOR P=1 TO N: FOR Q=6 TO I+7
1170 IF V[P]<=VAL(Z$(Q)) THEN D[2,Q-5]=D[2,Q-5]+1: GOTO 1190
1180 D[2,I+3]=0: NEXT Q
1190 NEXT P
1200 FOR P=1 TO I+2
1210 D[1,P]=VAL(Z$(P+5)): IF D[2,P]>=D2MAX THEN D2MAX=D[2,P]
1220 NEXT P
1230 DIM A[2,I+4]: FOR P=2 TO I+4
1240 A[1,P]=((D[1,P-1]-XMIN)*251/XR+117)*COS(0.32175)
1250 A[2,P]=A[1,P]*TAN(0.32175): NEXT
1260 RETURN
2000 FOR P=2 TO I+3: Q=0
2010 IF A[1,P]+Q>A[1,P+1] THEN 2090
2020 IF P<=I+2 THEN PLOT 112+A[1,P]+Q,34+A[2,P]+Q*TAN(0.3217
5)
2030 DRAWR 0,100/D2MAX*D[2,P]
2040 IF D[2,P]>=D[2,P-1] THEN DRAWR -50,0: GOTO 2070
2050 IF P<I+3 AND Q<50 AND Q/3<(D[2,P-1]-D[2,P])*100/D2MAX T
HEN DRAWR -Q,0: GOTO 2060 ELSE DRAWR -50,0
2060 IF D[2,P]<=D[2,P-1] THEN GOTO 2080
2070 IF Q=0 THEN DRAWR 0,-100/D2MAX*(D[2,P]-D[2,P-1])
2080 Q=Q+3: GOTO 2010
2090 NEXT
2100 PLOT 112,34: DRAWR 529,529*TAN(0.32175)
2110 PLOT 112,34: DRAWR -110,0
2120 PLOT 112+A[1,2],34+A[2,2]: DRAWR -50,0
2130 FOR P=1 TO I+1: LOCATE 1,3+P: PRINT LEFT$(Z$(P+5),5);"-
";LEFT$(Z$(P+6),5);" "D[2,P+1]: NEXT
2140 LOCATE 25,4: PRINT "Sigma=";LEFT$(z$(4),7)
2150 LOCATE 26,5: PRINT "Mean=";LEFT$(Z$(1),7)
2160 LOCATE 18,21: PRINT LEFT$(Z$(2),5)
2170 LOCATE 33,16: PRINT LEFT$(Z$(3),5)
2180 PLOT 224,68: DRAWR 50,0
2190 PLOT 464,148: DRAWR 50,0
2200 PLOT 112+((MEAN-XMIN)*255/XR+112)*COS(0.32175),34+((MEA
N-XMIN)*255/XR+112)*SIN(0.32175): DRAWR 0,200
2210 LOCATE 11,1: PRINT "STATISTICAL ANAYSIS"
2220 IF INKEY$="" THEN 2220
2230 RETURN
```

EXAMPLE

The following data give the display as shown.

V1=25.6	V15=35.0	V29=27.9	V43=28.9
V2=28.3	V16=27.1	V30=32.1	V44=32.0
V3=30.1	V17=29.3	V31=28.7	V45=27.9
V4=26.9	V18=30.6	V32=29.1	V46=31.0
V5=37.6	V19=30.5	V33=32.1	V47=30.5
V6=30.8	V20=29.5	V34=31.3	V48=29.9
V7=26.0	V21=28.0	V35=30.0	V49=30.1
V8=29.3	V22=33.1	V36=29.6	V50=30.3
V9=30.2	V23=36.0	V37=28.7	V51=28.7
V10=31.6	V24=29.8	V38=35.0	V52=31.2
V11=28.7	V25=31.2	V39=31.2	V53=33.1
V12=29.5	V26=30.1	V40=27.9	V54=28.9
V13=30.3	V27=28.7	V41=30.5	V55=27.9
V14=32.4	V28=31.5	V42=31.6	V56=31.0



86. STRING STORAGE

Storing data, particularly as floating point numbers can use a lot of memory and it is sometimes preferable to store as a string or string array.

There are three cases to consider. Firstly where each string is the same length as for instance with machine code bytes in hexadecimal. Here no separators are required. If however the data are different lengths then separators such as commas are needed. Where most of the strings are the same length but with a few smaller ones it may be economic to fill out with spaces and dispense with the separators.

The first case is trivial but the following two programs might be of use in saving memory.

```
10 REM All strings the same length
20 INPUT "Length of strings";N: B$=""
30 INPUT "String";A$: IF LEN(A$)>N THEN 30: REM To finish IN
PUT "="
40 IF A$<>"=" THEN B$=B$+STRING$(N-LEN(A$),32)+A$: GOTO 30
50 GOSUB 2000: END
2000 PRINT B$
2010 RETURN
2020 REM Changing line 40 to
2030 REM "IF A$<>"=" THEN B$=B$+STRING$(N-LEN(A$),32)+A$+NL$
: GOTO 30" (Where NL$=CHR$(10)+CHR$(13))
2040 REM PRINTs out in a vertical column.
2050 REM Or changing 2000 to
2060 REM "FOR P=1 TO T: PRINT MID$(B$,1+N*(P-1),N): NEXT" (W
here T is the number of strings) does the same.
```

```
10 REM Store as a string with a / between each item
20 DEFSTR N
30 INPUT N: REM N=n1/n2/n3/n4/n5/n6/ and must finish with a
slash (/). N.B. Commas are NOT allowed and inverted commas a
re not necessary.
40 GOSUB 2000: END
2000 REM To recover the data
2010 A=1: B=1
2020 FOR P=A TO LEN(N$)
2030 IF MID$(N$,P,1)="/" THEN A=P: PRINT MID$(N$,B+(B=2),A-B
-(B=2))
2040 B=A+1: NEXT
```

87. TAG PRINT

In MODE 2, PRINT produces the ASCII characters within a rectangular area 8x8 pixels in size which is quite difficult to read. By TAGging the PRINT to the graphics cursor it is possible to leave a little more space between the letters to suit your preference.

In MODE 1, the reverse can be used i.e. to have slightly less space between the letters.

```
10 REM TAG PRINT-MODE 1
20 MODE 1
30 INPUT "Printing position (as x,y coordinates)"; X,Y
40 INPUT "Matter to be printed";A$
50 GOSUB 2000: END
2000 CLS: FOR P=1 TO LEN (A$): Z$=MID$(A$,P,1)
2010 MOVE X,Y: TAG
2020 PRINT Z$;: X=X+14+5*(Z$=" ")
2030 IF X>620 THEN X=0: Y=Y-16
2040 NEXT: TAGOFF
2050 RETURN
```

```
10 REM TAG PRINT-MODE 2
20 MODE 2
30 INPUT "Printing position (as x,y coordinates)"; X,Y
40 INPUT "Matter to be printed";A$
50 GOSUB 2000: END
2000 CLS: FOR P=1 TO LEN (A$): Z$=MID$(A$,P,1)
2010 MOVE X,Y: TAG
2020 PRINT Z$;: X=X+10+4*(Z$=" ")
2030 IF X>620 THEN X=0: Y=Y-16
2040 NEXT: TAGOFF
2050 RETURN
```

88. TEST FOR A BINARY NUMBER

This test is similar to the test for a decimal number but more restricted. This subroutine tests for a positive or negative binary integer and can be used for example prior to a conversion routine.

```
10 REM Test for a binary number
20 PRINT "Type in the number as a string": PRINT
30 INPUT B$
40 GOSUB 1000: GOSUB 2000: END
1000 FAIL=0: T=1: IF LEFT$(B$,1)="+" OR LEFT$(B$,1)="-" THEN
 T=2
1010 IF T=2 AND LEN(B$)=1 THEN FAIL=-1: RETURN
1020 FOR P=T TO LEN(B$)
1030 IF MID$(B$,P,1)<>"0" AND MID$(B$,P,1)<>"1" THEN FAIL=-1
: RETURN ELSE NEXT
1040 RETURN
2000 IF NOT FAIL THEN PRINT "OK": RETURN
2010 PRINT "Not a binary number": RETURN
```

89. TEST FOR A DECIMAL NUMBER

As the INPUT is in the form of a string each character can be tested separately. "+" or "−" must come at the beginning if they are used and this changes T from 1 to 2 so that they are not involved in the test loop. S counts the number of decimal points and fails the number if it has more than one or the characters do not lie between 0 and 9. Lines 1010 and 1020 fail entries without digits.

The subroutine only passes positive or negative decimal numbers.

```
10 REM Test for a decimal number
30 INPUT "Type in the number";N$
40 GOSUB 1000: GOSUB 2000: END
1000 FAIL=0: S=0: T=1: IF LEFT$(N$,1)="+" OR LEFT$(N$,1)="-"
 THEN T=2
1010 SP=INSTR(N$,"."): IF SP=LEN(N$) THEN FAIL=-1: RETURN
1020 IF T=2 AND LEN(N$)=1 THEN FAIL=-1: RETURN
1030 FOR P=T TO LEN(N$): Z$=MID$(N$,P,1)
1040 IF Z$="." THEN S=S+1
1050 IF Z$="." OR Z$>="0" AND Z$<="9" THEN NEXT: IF S<2 THEN
 RETURN
1060 FAIL=-1
1070 RETURN
2000 IF NOT FAIL THEN PRINT N$: RETURN
2010 PRINT "Not a decimal number": RETURN
```

90. TIMER

To measure the efficiency of a program in terms of its execution time is very easy with the machine variable TIME.

The first statement should be placed after any INPUT statement and the second one after the program has finished calculating or PRINTing.

To illustrate its use, the actual delay time of an empty loop is measured.

```
10 REM Timer
20 INPUT N
30 T0=TIME
40 FOR P=1 TO N: NEXT: T=(TIME-T0)/300
50 PRINT "Time for";
60 PRINT N;
70 PRINT "loops is";
80 PRINT ROUND(T,2);"seconds"
```

EXAMPLE

```
run
? 10000
Time for 10000 loops is 10.71 seconds
```

91. UNDERLINE

To use the underline symbol on the keyboard requires an OR operation performed through the transparent printing option–PRINT CHR$(22)+CHR$(1) followed by CHR$(22)+CHR$(0) to disable it. However, the result is unsatisfactory as the underline is the bottom row of pixels and it merges with the letters.

Two alternatives produce a better result. TAG PRINT will position the word and the underline as required or the following subroutine which uses a modified CHR$(208), also looks well.

CHR$(208) has the top two lines of pixels set and line 1000 removes one of them. Line 1010 adds a second underline if required. SYMBOL AFTER 208 puts the pixel information for CHR$(208) onwards immediately after HIMEM.

```
10 REM Underline
20 SYMBOL AFTER 208: H=HIMEM+2
30 INPUT "Matter to be underlined";A$
40 PRINT "Printing position": PRINT: INPUT "Row No.";R: PRIN
T: INPUT "Column No.";C
50 GOSUB 1000: GOSUB 2000: END
1000 POKE H,0: RETURN
1010 REM To double underline, also POKE H+1,255
2000 CLS: LOCATE C,R: PRINT A$
2010 LOCATE C,R+1: FOR P=1 TO LEN(A$): PRINT CHR$(208);: NEX
T
2020 REM To PRINT on the line immediately below the the unde
rlined words, use PRINT CHR$(22);CHR$(1); to enable the tran
sparent option and CHR$(22);CHR$(0); to disable the option w
hen no longer required.
2030 REM Alternatively, protect the underline with a PRINT s
tatement at the end of line 2010 if no printing appears imme
diately under the words.
2040 RETURN
```

92. UNIVERSAL ROTATION

This subroutine allows any set of points (which could represent a plane or solid figure) to be rotated by some angle theta about any line in space which passes through two points (AX,AY,AZ) and (BX,BY,BZ). In the illustration, a bipyramid is rotated 30° about a line passing through the centre of the figure and a point (1,1,1) i.e. a line which emerges from the centre of a triangular face.
In the input part of the program, the coordinates (AX,AY,AZ) and (BX,BY,BZ) are entered and then the 3–D coordinates of the corners of the solid figure are INPUT but changed before storing in C[N,2] so that the origin is moved to (AX,AY,AZ). Theta is entered in degrees.

In the main program. the x,y,z displacements DX,DY,DZ between the two points on the line are calculated and the program then goes to SUB 1100 which sets up a 3x3 Unit Matrix. This is used in SUB 1110 to build up the main transformation matrix via SUB 1200 and SUB 1300. SUB 1200 works out the angle for the various rotations and SUB 1300 first finds the correct rotation matrix A[3,3] which are as follows for the three axes:–

$$\underset{LI=1}{\begin{bmatrix}1 & 0 & 0\\ 0 & \cos A & -\sin A\\ 0 & \sin A & \cos A\end{bmatrix}} \quad \underset{LI=2}{\begin{bmatrix}\cos A & 0 & \sin A\\ 0 & 1 & 0\\ -\sin A & 0 & \cos A\end{bmatrix}} \quad \underset{LI=3}{\begin{bmatrix}\cos A & -\sin A & 0\\ \sin A & \cos A & 0\\ 0 & 0 & 1\end{bmatrix}}$$

The latter part of the routine multiplies A by U and then transfers the result back to U leaving A and Z available for the next multiplication.

Referring to the sequence of diagrams for the transformation, these perform the following operations:–

1. Move the origin to (AX,AY,AZ). This is done in the INPUT
2. Rotate about the z–axis to put B in the x–z plane. LI=3; angle= ATN(DY/DX);A=–angle;B stores A
3. Rotate about the y–axis to make AB coincide with the z–axis. LI=2;angle=ATN(SQR(DX^2+DY^2)/DZ);A=–angle;C stores A
4. Rotate theta degrees about the z–axis. LI=3;A=THETA
5. Rotate back about the y–axis by angle C. LI=2;A=C
6. Rotate back about the z–axis by angle B. LI=3;A=B

We now have the main transformation matrix which is the product of six matrices i.e. $A1^{-1}$ *$A2^{-1}$ *A3*A2*A1*U where A1, A2 and A3 are the separate rotation matrices and $A1^{-1}$ and $A2^{-1}$ the inverses of A1 and A2 which rotate in the opposite direction. Note that it is not necessary to

formally invert as we know that changing the sign of the angle must reverse the rotation process. However it is easy to show that

$$\begin{bmatrix} \cos(-A) & 0 & \sin(-A) \\ 0 & 1 & 0 \\ -\sin(-A) & 0 & \cos(-A) \end{bmatrix} * \begin{bmatrix} \cos(A) & 0 & \sin(A) \\ 0 & 1 & 0 \\ -\sin(A) & 0 & \cos(A) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

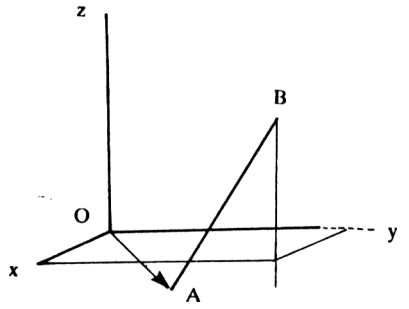
as $\cos^2 A + \sin^2 A = 1$

After returning from SUB 1110, T[N,2] is made available to store the transformed coordinates which are produced by multiplying C by U and adding AX,AY,AZ to x, y and z to move the origin back to where it started.

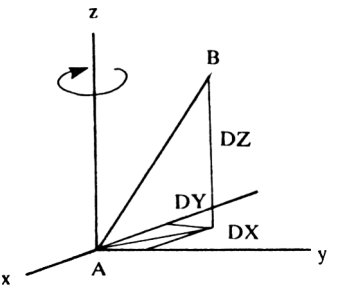
In the output, a magnification factor can be incorporated and the PLOT and DRAW information is INPUT into V$[N,N] to enable the figure to be drawn after using the projection formula and allowing for the screen aspect ratio. To condense the information, only the upper right triangle of V$[N,N] is used but note line 2150 – NEXT Q: NEXT P – NEXT Q,P does not work.

```
10 REM Universal rotation
20 DEG: DEF FN A$(A)=MID$(STR$(A),2)
30 CLS: PRINT "      Type in the coordinates of the two point
s which define the axes of rotation"
40 INPUT "AX";AX: LOCATE 13,VPOS(#0)-1: INPUT "AY";AY: LOCAT
E 26,VPOS(#0)-1: INPUT "AZ";AZ
50 INPUT "BX";BX: LOCATE 13,VPOS(#0)-1: INPUT "BY";BY: LOCAT
E 26,VPOS(#0)-1: INPUT "BZ";BZ
60 INPUT "Number of points";N: IF N<=0 THEN 60
70 DIM C[N,2],T[N,2]: CLS
80 PRINT "   Type in the x, y and z coordinates of the point
s to be rotated"
90 FOR P=1 TO N
100 LOCATE 1,P+3: PRINT "x";FN A$(P);: INPUT Z: C[P,0]=Z-AX
110 LOCATE 14,P+3: PRINT "y";FN A$(P);: INPUT Z: C[P,1]=Z-AY
120 LOCATE 27,P+3: PRINT "z";FN A$(P);: INPUT Z: C[P,2]=Z-AZ
130 NEXT
140 INPUT "      OK?     (Y/N) ";K$: K$=UPPER$(K$): IF K$="Y"
 THEN CLS ELSE ERASE C,T: GOTO 60
150 INPUT " Type in the desired angle of rotation, theta in
degrees";TH
160 GOSUB 1000: GOSUB 2000: END
1000 DX=BX-AX: DY=BY-AY: DZ=BZ-AZ
1010 GOSUB 1100: GOSUB 1110
1020 FOR P=1 TO N
1030 T[P,0]=C[P,0]*U[0,0]+C[P,1]*U[0,1]+C[P,2]*U[0,2]+AX
1040 T[P,1]=C[P,0]*U[1,0]+C[P,1]*U[1,1]+C[P,2]*U[1,2]+AY
1050 T[P,2]=C[P,0]*U[2,0]+C[P,1]*U[2,1]+C[P,2]*U[2,2]+AZ
1060 NEXT: ERASE U
1070 RETURN
```

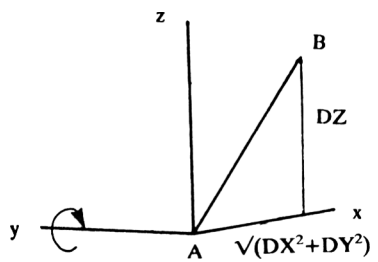
```
1099 REM Unit matrix
1100 DIM U[2,2]: FOR I=0 TO 2: U[I,I]=1: NEXT :RETURN
1109 REM Universal Rotation
1110 GOSUB 1200
1120 LI=2: B=A: A=-A: GOSUB 1300
1130 DY=SQR(DX*DX+DY*DY): DX=DZ: GOSUB 1200
1140 LI=1: C=A: A=-A: GOSUB 1300
1150 LI=2: A=TH: GOSUB 1300
1160 LI=1: A=C: GOSUB 1300
1170 LI=2: A=B: GOSUB 1300
1180 RETURN
1199 REM Angle
1200 IF ABS(DX)>0.000001 THEN 1240 ELSE A=90
1210 IF DY<0 THEN A=A+90
1220 IF ABS(DY)<0.000001 THEN A=0
1230 RETURN
1240 A=ATN(DY/DX): IF DX<0 THEN A=A+180
1250 RETURN
1299 REM Turn and Multiply
1300 DIM A[2,2]: A[LI,LI]=1
1310 A1=(LI+1)MOD 3: A2=(LI+2)MOD 3
1320 CA=COS(A): SA=SIN(A)
1330 A[A1,A1]=CA: A[A2,A2]=CA: A[A1,A2]=-SA: A[A2,A1]=SA
1340 FOR I=0 TO 2: DIM Z[2]: FOR J=0 TO 2: FOR K=0 TO 2
1350 Z[J]=Z[J]+A[J,K]*U[K,I]: NEXT K,J
1360 FOR K=0 TO 2: U[K,I]=Z[K]: NEXT
1370 ERASE Z: NEXT: ERASE A: RETURN
2000 FOR P=1 TO N
2010 PRINT T[P,0];T[P,1];T[P,2]
2020 NEXT
2099 REM Information needed for graphical output as in examp
le
2100 INPUT "Magnification";H
2110 DIM V$[N,N]
2120 PRINT "Type in "+CHR$(34)+"1"+CHR$(34)+" to represent v
ertices to be joined else a "+CHR$(34)+"0"+CHR$(34)
2130 FOR P=1 TO N: FOR Q=P+1 TO N
2140 IF P<>Q THEN PRINT "Vertex";P;"joined to vertex ";Q;: I
NPUT V$(P,Q)
2150 NEXT Q: NEXT P
2160 GOSUB 2500
2170 RETURN
2499 REM Graphic output
2500 CLS: DIM A[N,1]
2510 FOR P=1 TO N
2520 A[P,0]=320-(58.8*T[P,0]-176.4*T[P,1])*H
2530 A[P,1]=196-(19*T[P,0]+6.5*T[P,1]-162*T[P,2])*H
2540 NEXT
2550 FOR P=1 TO N: FOR Q=P TO N
2560 IF V$[P,Q]="1" THEN PLOT A[P,0],A[P,1]: DRAW A[Q,0],A[Q
,1]
2570 NEXT Q,P
2580 ERASE A,V$
2590 RETURN
```



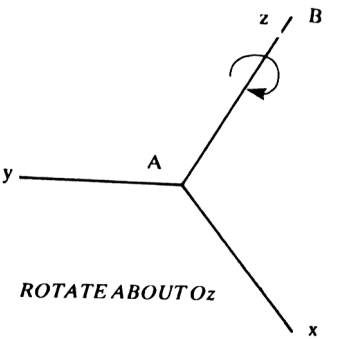
1. MOVE THE ORIGIN TO A



2. ROTATE ABOUT Oz TO PUT AB IN THE x−z PLANE

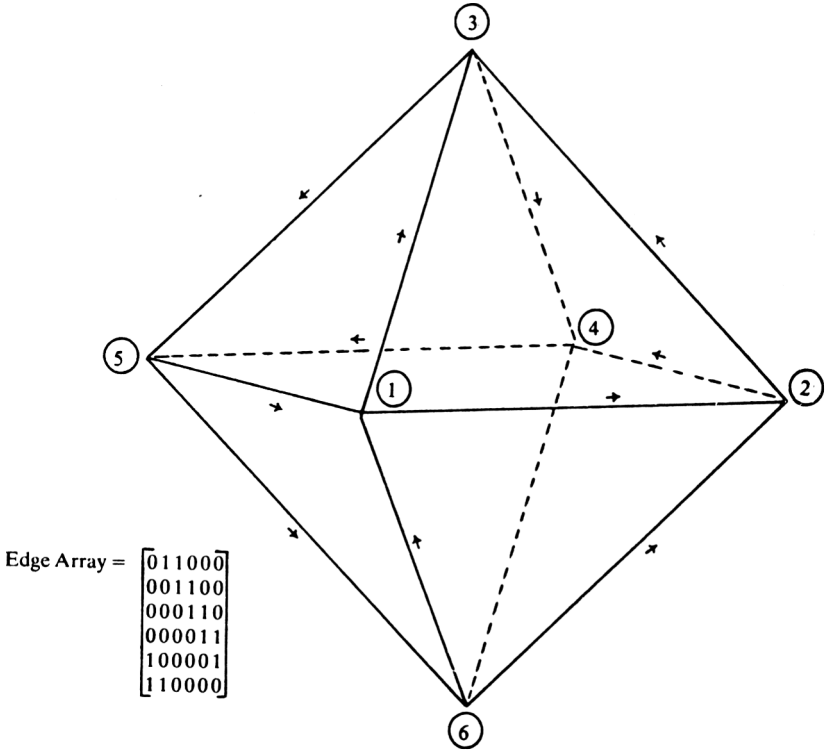


3. ROTATE ABOUT Oy TO MAKE Oz COINCIDE WITH AB



4. ROTATE ABOUT Oz

Fig. 92.1 Making the Z Axis Coincide with the Line



Edge Array = $\begin{bmatrix} 011000 \\ 001100 \\ 000110 \\ 000011 \\ 100001 \\ 110000 \end{bmatrix}$

Fig. 92.2 Use of Arrays to Hold Edge or Path Information

93. USEFUL FUNCTIONS

It saves space in a program to use a DEF FN if the function is needed more than once, the actual saving depending on the complexity of the function.

The functions listed here are generally useful for a range of programs and can be found in various parts of the book.

```
10 REM PEEK (into a two byte address)
20 DEF FN P(P)=PEEK(P)+256*PEEK(P+1)
30 REM POKE (a number into a two byte location)
40 DEF FN Q(Q)=Q-256*INT(Q/256): REM low byte
50 DEF FN R(R)=INT(Q/256): REM high byte
60 REM Larger of two numbers
70 DEF FN L(A,B)=A-(B-A)*(B>A)
80 REM or
90 DEF FN L(A,B)=(A+B+ABS(A-B))/2
100 REM Erase and backspace N characters
110 DEF FN BS$(N)=SPACE$(N)+STRING$(N,CHR$(8))
120 REM To PRINT or not to PRINT based on the result of a lo
gical operation
130 DEF FN FB A$(A$,BOOLE)=MID$(A$,1,-LEN(A$)*BOOLE)
140 REM Plus or minus PRINT for "+" or "-" on a string expre
ssion
150 DEF FN SIGN$(X)=CHR$(45+2*(X>=0))
160 REM Packed format PRINT-omits sign space in front of num
bers
170 DEF FN A$(A)=MID$(STR$(A),2)
180 REM Packed format PRINT, positive-omit space in front of
 positive numbers only
190 DEF FN A$(A)=MID$(STR$(A),1-LEFT$(STR$(A),1)<>"-",6): RE
M 6 digits
200 REM Complex numbers-enables complex numbers to be PRINTe
d correctly
210 DEF FN A$(A$,A,BOOLE)=MID$(A$,1+A,-LEN(A$)*BOOLE)
220 REM Brackets-for PRINTing in certain circumstances e.g.
around negative numbers
230 DEF FN L$(BOOLE)=CHR$(32-8*BOOLE): DEF FN R$(BOOLE)=CHR$
(32-9*BOOLE)
```

Full Name	*Function*	*Derived Function*
Secant	SEC	1/COS(X)
Cosecant	CSC	1/SIN(X)
Cotangent	COT	1/TAN(X)
Inverse Sine	ARCSIN*	ATN(X/SQR(1-X*X))
Inverse Cosine	ARCCOS*	-ATN(X/SQR(1-X*X))+1.5708
Inverse Secant	ARCSEC	ATN(X/SQR(X*X-1))+SGN(SGN(X)-1)*1.5708
Inverse Cosecant	ARCCSC	ATN(X/SQR(X*X-1))+(SGN(X)-1)*1.5708
Inverse Cotangent	ARCCOT	ATN(X)+1.5708
Hyperbolic Sine	SINH*	(EXP(X)-EXP(-X))/2
Hyperbolic Cosine	COSH*	(EXP(X)+EXP(-X))/2
Hyperbolic Tangent	TANH*	(EXP(X)-EXP(-X))/(EXP(X)+EXP(-X))
Hyperbolic Secant	SECH	2/(EXP(X)+EXP(-X))
Hyperbolic Cosecant	CSCH	2/(EXP(X)-EXP(-X))
Hyperbolic Cotangent	COTH	(EXP(X)+EXP(-X))/(EXP(X)-EXP(-X))
Inverse Hyperbolic Sine	ARCSINH*	LOG(X+SQR(X*X+1))
Inverse Hyperbolic Cosine	ARCCOSH*	LOG(X+SQR(X*X-1))
Inverse Hyperbolic Tangent	ARCTANH*	LOG((1+X)/(1-X))/2
Inverse Hyperbolic Secant	ARCSECH	LOG((SQR(1-X*X)+1)/X)
Inverse Hyperbolic Cosecant	ARCCSCH	LOG((SGN(X)*SQR(1+X*X)+1)/X)
Inverse Hyperbolic Cotangent	ARCCOTH	LOG((X+1)/(X-1))/2

* Contains a Reserved Name (OK for Amstrad but not other computers)

N.B. 1.5708 = PI/2

Table 93.1: Trigonometric Functions for use in DEF FN statements

The Routines

The following routines are complete programs, ready for you to type directly into your Amstrad. Whilst being of interest in their own right, they also demonstrate the uses of several of the subroutines listed in this book.

1. ANAGRAM

This program gives all the anagrams from a word taking into account any known letters in the correct solution.

The program first seeks any known letters and eliminates them from C$ (lines 140–200). If the letters in the original word and the known letters do not match up, the Q loop is exceeded at 190 and the program restarts.

C$ is permuted by lines 220–360. The permutations are PRINTed in line 260 or, if some letters are known, in line 290 after restoring these letters as only the unknown ones are permuted.

```
10 REM Anagram
20 DEFINT C,L,P,X,Z: NL$=CHR$(10)+CHR$(13)
30 CLS: Z=0: LINE INPUT "Type in the word  ";A$: L=LEN(A$):
GOSUB 400: GOSUB 370: IF Z=1 THEN 30
40 B$="": C$=A$
50 PRINT NL$;"Do you know the positions of any of the letter
s?";NL$: INPUT "(y/n)";K$
60 K$=UPPER$(K$): IF K$="N" THEN 220
70 CLS: LOCATE 2,5: PRINT A$: LOCATE 1,8: PRINT "  Type in t
he known letters of the word one by one in the correct place
 or type a question mark if the letter position  is not know
n.";NL$;NL$;CHR$(32);STRING$(L,"*")
80 FOR P=1 TO L
90 LOCATE P+1,20: LINE INPUT X$: IF X$="" THEN 90
100 B$=B$+LEFT$(X$,1): LOCATE P+1,12
110 IF LEFT$(X$,1)="?" THEN PRINT "?";: GOTO 130
120 PRINT LEFT$(X$,1)
130 NEXT P: GOSUB 370: D$=B$: CLS: IF Z=1 THEN 30
140 FOR P=1 TO L
150 IF MID$(B$,P,1)="?" THEN 200
160 FOR Q=1 TO LEN(C$)
170 IF MID$(B$,P,1)<>MID$(C$,Q,1) THEN 190
180 C$=LEFT$(C$,Q-1)+MID$(C$,Q+1): GOTO 200
190 NEXT Q: IF Q=LEN(C$)+1 THEN 210
200 NEXT P: IF LEN(C$)=0 THEN PRINT A$: END ELSE GOTO 220
```

```
210 PRINT NL$; "Your information is inconsistent. Press any
key to start again": GOSUB 350: CLS: GOTO 30
220 C=LEN(C$)-1: DIM A[C],B[C]
230 FOR P=1 TO C: A[P]=1: B[P]=C-P+2: NEXT
240 FOR P=1 TO C: IF A[P]>B[P] THEN A[P]=1
250 NEXT P: T=1
260 IF K$="N" OR K$="n" THEN PRINT C$+SPACE$(7-C);: GOTO 300
270 FOR P=1 TO L
280 IF MID$(B$,P,1)="?" THEN MID$(D$,P,1)=MID$(C$,T,1): T=T+
1
290 NEXT P: PRINT D$+SPACE$(8-L);
300 FOR X=C TO 1 STEP -1
310 Z$=LEFT$(C$,1): FOR P=1 TO 1+C-X
320 MID$(C$,P,1)=MID$(C$,P+1,1): NEXT P
330 MID$(C$,P,1)=Z$: A[X]=A[X]+1
340 IF A[X]>B[X] THEN NEXT X: END ELSE 240
350 IF INKEY$="" THEN 350
360 RETURN
370 PRINT NL$;NL$;"     Is this OK? (y/n)";: INPUT K$
380 K$=UPPER$(K$): IF K$="Y" THEN RETURN
390 Z=1: RETURN
400 FOR P=1 TO L: Z$=MID$(A$,P,1)
410 IF (Z$<"A" OR Z$>"Z") AND (Z$<"a" OR Z$>"z") THEN Z=1: P
=L
420 NEXT P: RETURN
```

EXAMPLE

```
run
Type in the word? easter
Is this ok? (y/n)? y
Do you know the position of any of the letters? (y/n)? y

easter

Type in the word again using the known letters and question marks

******

s????r

Is this ok? (y/n) y

seater  saeter  sateer  staeer  steaer
setaer  sateer  staeer  steaer  setaer
seater  saeter  steear  setear  seetar
seetar  setear  steear  seeatr  seeatr
seaetr  saeetr  saeetr  seaetr
Ready
```

2. CONTROL LOOP SIMULATION

In automatic control, there is a sensor which measures the quantity you are trying to control and if the measurement differs from the ideal value some change is made to bring the measurement back to the ideal value. Normally the rate of correction is made to be proportional to the error signal but integral and derivative terms can be added to improve the control action.

A time delay in a control loop can play havoc with its stability and lead to hunting or run away conditions. Backlash in transmission mechanisms or a time interval between the measuring and control point are common sources of this problem.

In the following program, the control stability of a machining operation is simulated to find the maximum rate of correction to give a stable loop.

As it is not possible to measure at the point of machining the size measurement is taken diametrically opposite to the tool position and this introduces a half cycle delay into the control loop.

To test the stability, a step change is introduced into the system and its response calculated. In practice this could be a bit of the tool breaking off or the setting of the tool being accidentally altered but slow changes can be considered as a set of small step changes.

For the first half of the cycle no change in measurement occurs as the new size has not yet got to the measurement point. When it arrives, it remains a constant error for a further half cycle and the control system institutes a linear correction. When this corrected size arrives the control changes to a squared function then to a cubic one and so on. Whether or not this is stable depends on the amount of correction done in the second half cycle after the step change i.e. whether it has over corrected or under corrected.

To investigate this you simply need to change K in line 60 and observe the graph of tool position versus time. This can vary from over damped (K=.01) through critically damped (K=.018) to wildly oscillating (K=0.1)

The simulation indicates that the system works best if only 37% of the error is corrected in one half cycle (K=.018).

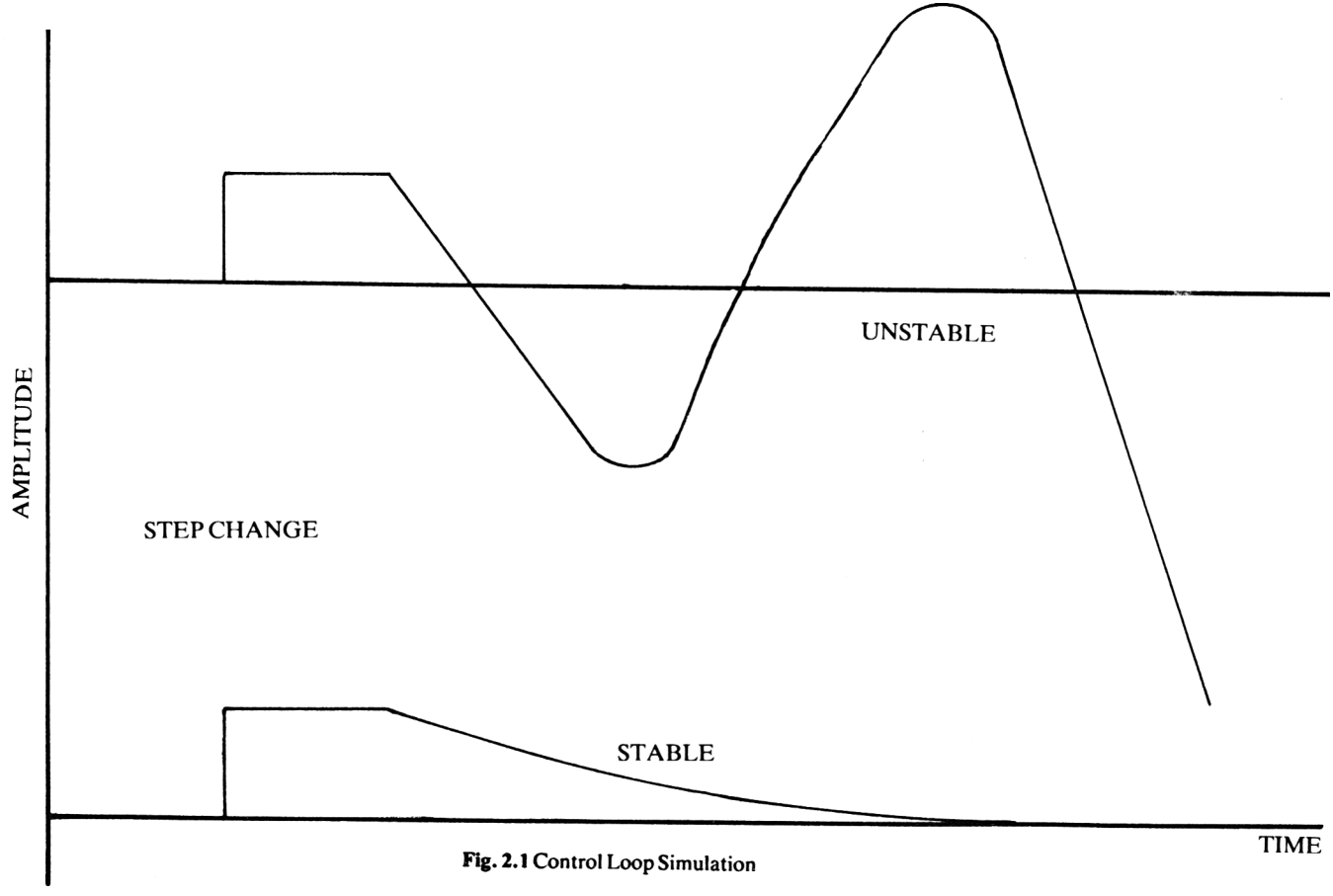


Fig. 2.1 Control Loop Simulation

```
10 REM Control loop stability
20 MODE 1: BORDER 1: INK 0,15: INK 1,9: INK 2,0: INK 3,6
30 SYMBOL AFTER 208: POKE HIMEM+1,0
40 DEF FN H(T,X,Y)=-(X<=T)+(Y<=T): REM Heaviside operator
50 CLS: PEN 2: PAPER 3: PRINT SPC(9);"Control Loop Stability
"; SPC(9): PRINT SPC(9);: FOR P=1 TO 22: PRINT CHR$(208);: N
EXT: PRINT SPC(9): PRINT
60 PAPER 0: PRINT " This program demonstrates the behaviouro
f a control loop which has a time delayin it so that the rat
e of correction of the error determines the stability of   t
he system."; SPC(29)
70 PRINT "   By inserting different values of the proportion
ality constant K, you can see how this affects the response
to a step change. K should be in the range 0-0.1  to show th
e full effect."; SPC(16)
80 PAPER 3: PRINT: PRINT " Proportionality constant": PRINT:
 INPUT "K";K
90 RO=100: DR=20: W=0.1: T1=PI/W: Z=K*T1
100 CLS
110 PRINT: PRINT SPC(11);"Initial Conditions": PRINT SPC(11)
;: FOR P=1 TO 18: PRINT CHR$(208);: NEXT
120 PAPER 0: PRINT: PRINT TAB(3);"Radius";TAB(16) USING "###
.####";RO
130 PRINT: PRINT TAB(3);"Step";TAB(16) USING "###.####";DR
140 PRINT: PRINT TAB(3);"Half Cycle";TAB(16) USING "###.####
";T1
150 PRINT: PRINT TAB(3);"Constant K";TAB(16) USING "###.####
";K
160 PRINT: PRINT "       OK?  (Y/N)";: INPUT K$: K$=UPPER$(K$
): IF K$<>"Y" THEN CLS: GOTO 80
170 T2=T1+T1: T3=T2+T1: T4=T3+T1: T5=T4+T1: T6=T5+T1: T7=R6+
T1: REM Half cycle periods
180 TAG: MOVE 576,47: PRINT "AXIS": CLS : TAGOFF
190 FOR T=1 TO T3
200 R=RO+FN H(T,T1,T2)*DR+FN H(T,T2,T3)*DR*(1-K*(T-T2))
210 GOSUB 340: NEXT T
220 FOR T=T3 TO T4: C=K*(T-T3)
230 R=RO+DR*(1-Z-C+C^2/2)
240 GOSUB 340: NEXT T
250 FOR T=T4 TO T5: C=K*(T-T4)
260 R=RO+DR*(1-2*Z+Z^2/2-(1-Z)*C+C^2/2-C^3/6)
270 GOSUB 340: NEXT T
280 FOR T=T5 TO T6: C=K*(T-T5)
290 R=RO+DR*(1-3*Z+2*Z^2-Z^3/6-C*(1-2*Z+Z^2/2)+C^2*(1-Z)/2-C
^3/6+C^4/24)
300 GOSUB 340: NEXT T
310 FOR T=T6 TO T7: C=K*(T-T6)
320 R=RO+DR*(1-4*Z+9*Z^2/2-4+Z^3/3+Z^4/24-C*(1-3*Z+2*Z^2-Z^3
/6)+C^2*(1-2*Z+Z^2/2)/2-C^3*(1-Z)/6+C^4/24-C^5/12)
330 GOSUB 340: NEXT T: GOTO 350
340 PLOT 3*T,40: DRAWR 0,R: RETURN
350 LOCATE 1,24: PEN 2: END
```

3. CUBIC CRYSTALS

The faces of a crystal can be designated by three whole numbers h, k and l (which are the reciprocals of the intercepts of the particular face on the x, y and z axes). In the fully symmetrical crystal, the Millar Indices, as these numbers are called are positive and negative and fully permuted by the symmetry to give a group of faces which enclose the crystal shape.

Thus, (100) i.e. h, k and l become $(\bar{1}00)$, (010), $(0\bar{1}0)$, (001) and $(00\bar{1})$ which encloses a cube. (111) has eight permutations and gives a bipyramid, (110) has twelve and gives a rhombic dodecahedron etc.,etc.

The program first works out a control array C[26,3] which will enable the correct lines to be drawn and then asks for the values of h, k and l you wish to choose. It calculates the coordinates of the apices of the figure and joins them up in a 3−D projection using solid lines for the front and dotted lines for the back,

The figure is labelled {hkl} form and the next values can be inserted by initially pressing any key followed by the new h, k, and l values.

Lines 100−220 set up a 3x8x6 matrix containing all the permuations of h, k and l with both positive and negative values.

Lines 240−245 contain the necessary information to enable 26 triplets of faces to be selected from the 48. These are stored in A[26,3,3] via lines 250−290. SUB's 1200, 1210 and 1220 select parts of this array containing different values of H, K and L (lines 10, 20 and 30) to make the control array C[26,3].

After the first three cycles controlled by G, the desired h,k and l values are asked for (lines 40−43) and lines 50 and 60 put their absolute values in descending order.

To find an apex, three faces which join at the apex are chosen and their equations solved as simultaneous equations. This gives the coordinates of the apex as this is the only point common to all three planes.

This is done in lines 300−390 where each of the 26 triplets is solved for x, y and z provided that the determinant D, is not zero. The results are stored in I[26,3].

Lines 400−420 use the projection formula to find the 2−D coordinates on the screen. H is used as a scaling factor.

Lines 440–480 separate the faces into seven different types:–

h	k	l
h	l	l
h	h	l
h	h	h
h	k	0
h	h	0
h	0	0

Subroutines 500, 600, 700, 800, and 900 decide which points to join up on the basis of information in the control array. For example, in a cube only adjacent corners must be joined and not face or body diagonals. SUB 1500 is for solid lines and SUB 1600 for dotted lines.

To help you when you are typing in the program, the main flow chart and the crystal face separation chart are appended as well as the control array C[26,3] and the face triplet matrix A[26,3,3].

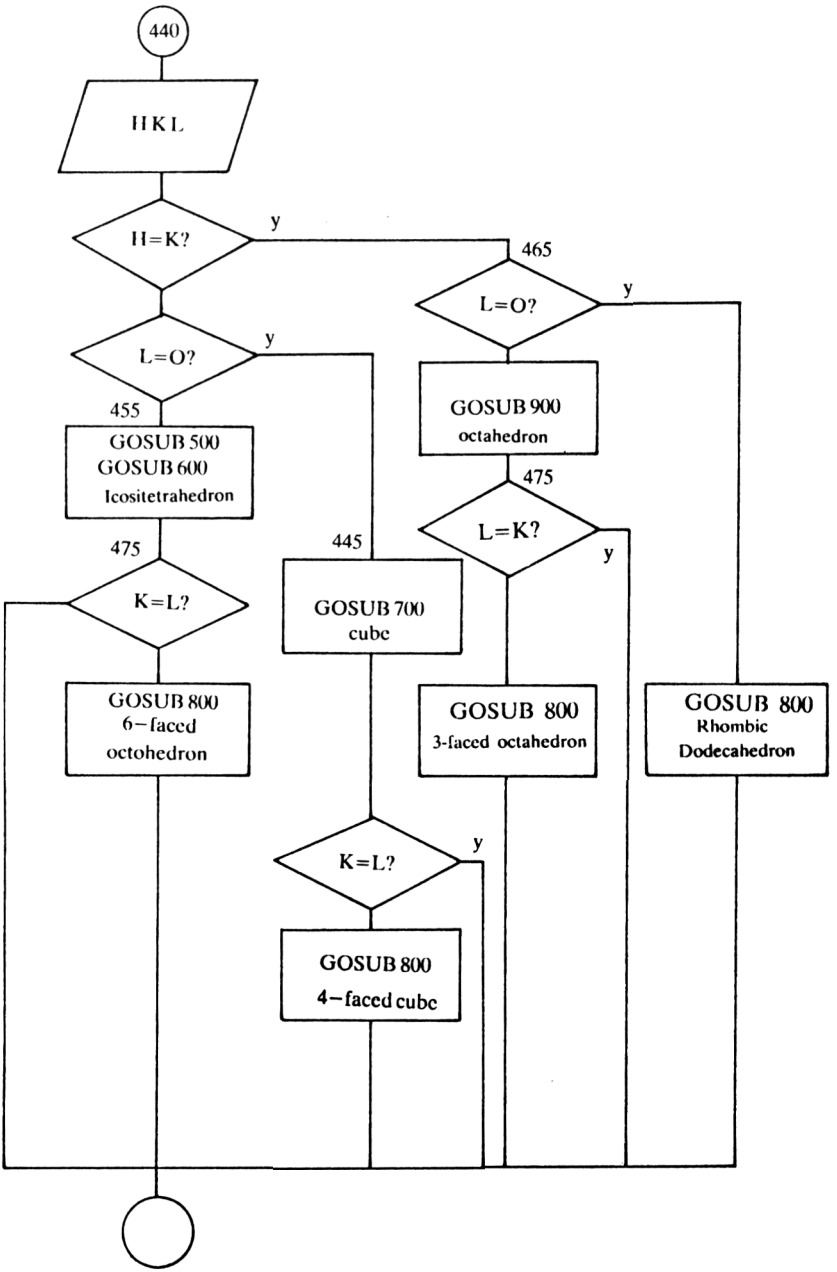
```
1 REM Cubic Crystal 4/m3m Class-Holosymmetric
2 BORDER 13: INK 0,2: INK 1,0
3 DEFINT B-D,G,H,K,L,N,R-Z: DEF FN A$(A)=MID$(STR$(A),2)
10 DIM C[26,3]: G=0: H=1: K=1: L=1: GOTO 100
20 ERASE A: G=G+1: IF G=1 THEN L=0: GOTO 100
30 IF G=2 THEN K=0: GOTO 100
40 PRINT "Next Form"
41 INPUT "h=";H: H=ABS(H)
42 INPUT "k=";K: K=ABS(K)
43 INPUT "l=";L: L=ABS(L): IF H+K+L=0 THEN 40
50 IF K>H THEN SW=K: K=H: H=SW
60 IF L>K THEN SW=L: L=K: K=SW: GOTO 50
70 Z$="CUBIC CRYSTAL FORM "+CHR$(123)+FN A$(H)+FN A$(K)+FN A
$(L)+CHR$(125)
75 CLS
100 DIM A[8,2],D[18],I[3,8,6]
110 X[0]=1: X[1]=1: X[2]=1
120 FOR A=1 TO 8: FOR B=0 TO 2: A[A,B]=X[B]: NEXT
130 IF A=2 OR A=6 THEN X[2]=-X[2]: GOTO 160
140 IF A=4 THEN X[0]=-X[0]: X[1]=-X[1]: X[2]=-X[2]: GOTO 160
150 X[1]=-X[1]
160 NEXT
170 X[0]=H: X[1]=K: X[2]=L: N=1
180 FOR B=0 TO 2: D[N]=X[B]: N=N+1: NEXT: SW=X[0]: X[0]=X[2]
: X[2]=SW
190 FOR B=0 TO 2: D[N]=-X[B]: N=N+1: NEXT: SW=X[0]: X[0]=X[1
]: X[1]=SW
200 IF N<18 THEN 180
210 FOR B=1 TO 8: N=1
220 FOR C=1 TO 6: FOR A=0 TO 2: I[A+1,B,C]=A[B,A]*D[N]: N=N+
1: NEXT A,C,B
```

```
230 ERASE A,D: DIM A[26,3,3]
240 A$="11864712345678111333555777"
241 B$="26357412345678888666444222"
242 C$="64711812345678426248862684"
243 D$="13613611111111135135135135"
244 E$="13613633333333642642642642"
245 F$="42542555555555135135135135"
250 FOR C=1 TO 26: FOR A=1 TO 3
260 A[C,1,A]=I[A,VAL(MID$(A$,C,1)),VAL(MID$(D$,C,1))]
270 A[C,2,A]=I[A,VAL(MID$(B$,C,1)),VAL(MID$(E$,C,1))]
280 A[C,3,A]=I[A,VAL(MID$(C$,C,1)),VAL(MID$(F$,C,1))]
290 NEXT A,C: ERASE I
291 IF G=0 THEN GOSUB 1200: GOTO 20
292 IF G=1 THEN GOSUB 1210: GOTO 20
293 IF G=2 THEN GOSUB 1220: GOTO 20
300 DIM I[26,3]: FOR A=1 TO 26
310 R=A[A,1,1]: S=A[A,1,2]: T=A[A,1,3]
320 U=A[A,2,1]: V=A[A,2,2]: W=A[A,2,3]
330 X=A[A,3,1]: Y=A[A,3,2]: Z=A[A,3,3]
340 D=R*(V*Z-Y*W)+S*(W*X-Z*U)+T*(U*Y-X*V)
350 IF D=0 THEN 390
360 I[A,1]=(V*Z-Y*W+S*(W-Z)+T*(Y-V))/D
370 I[A,2]=(R*(Z-W)+W*X-Z*U+T*(U-X))/D
380 I[A,3]=(R*(V-Y)+S*(X-U)+U*Y-X*V)/D
390 NEXT: ERASE A
400 DIM A[26,3]: FOR A=1 TO 26
410 A[A,1]=320-58.8*I[A,1]*H+176.4*I[A,2]*H
420 A[A,2]=196-19*I[A,1]-6.5*I[A,2]*H+162*I[A,3]*H: NEXT: ER
ASE I
430 MODE 2
440 IF H=K THEN 465 ELSE IF L<>0 THEN 455
445 GOSUB 700: IF K<>L THEN GOSUB 800
450 GOTO 2000
455 GOSUB 500: GOSUB 600: IF K<>L THEN GOSUB 800
460 GOTO 2000
465 IF L<>0 THEN GOSUB 900: GOTO 475
470 GOSUB 800: GOTO 2000
475 IF L<>K THEN GOSUB 800
480 GOTO 2000
500 FOR A=1 TO 6: FOR B=1 TO 3: IF C[A,B]<>0 THEN GOSUB 520
510 NEXT B,A: RETURN
520 FOR C=15 TO 26: IF C[C,B]<>C[A,B] THEN 550
530 IF C[C,1]=-1 OR A=4 THEN GOSUB 1600: GOTO 550
540 GOSUB 1500
550 NEXT C: RETURN
600 FOR A=7 TO 14: FOR C=15 TO 26
610 IF (C[C,1]=C[A,1] OR C[C,1]=0) AND (C[C,2]=C[A,2] OR C[C
,2]=0) AND (C[C,3]=C[A,3] OR C[C,3]=0) THEN GOSUB 630
620 NEXT C,A: RETURN
630 IF A>10 AND C>19 THEN GOSUB 1600: GOTO 650
640 GOSUB 1500
650 RETURN
700 FOR A=7 TO 14 STEP 2: FOR C=7 TO 14
```

```
710 IF C[A,1]=-C[C,1] AND C[A,2]=C[C,2] AND C[A,3]=C[C,3] OR
 C[A,1]=C[C,1] AND C[A,2]=-C[C,2] AND C[A,3]=C[C,3] OR C[A,1
]=C[C,1] AND C[A,2]=C[C,2] AND C[A,3]=-C[C,3] THEN GOSUB 730
720 NEXT C,A: RETURN
730 IF C=14 THEN GOSUB 1600: GOTO 750
740 GOSUB 1500
750 RETURN
800 FOR A=7 TO 14: FOR B=1 TO 3: FOR C=1 TO 6: IF C[A,B]=C[C
,B] THEN GOSUB 820
810 NEXT C,B,A: RETURN
820 IF A>10 THEN 840
830 GOSUB 1500: GOTO 860
840 IF C[A,2]=-1 OR C[C,1]=-1 THEN GOSUB 1600: GOTO 860
850 GOTO 830
860 RETURN
900 FOR A=1 TO 6: FOR B=1 TO 3: IF C[A,B]<>0 THEN GOSUB 920
910 NEXT B,A: RETURN
920 FOR C=1 TO 6: IF B=3 THEN 950
930 IF C[C,B+1]<>0 THEN 970
940 GOTO 990
950 IF C[C,1]<>0 THEN 970
960 GOTO 990
970 IF C=4 OR A=4 THEN GOSUB 1600: GOTO 990
980 GOSUB 1500
990 NEXT: RETURN
1200 FOR A=7 TO 14: GOSUB 1300: NEXT: RETURN
1210 FOR A=15 TO 26: GOSUB 1300: NEXT: RETURN
1220 FOR A=1 TO 6: GOSUB 1300: NEXT: RETURN
1300 FOR B=1 TO 3: C[A,B]=A[A,1,B]: NEXT: RETURN
1500 PLOT A[A,1],A[A,2]: DRAW A[C,1],A[C,2]: RETURN
1600 IF A[A,1]=A[C,1] THEN 1670 ELSE IF A[A,2]=A[C,2] THEN 1
630
1610 P=(A[C,2]-A[A,2])/(A[C,1]-A[A,1])
1620 IF ABS(P)<1 THEN 1640 ELSE P=1/P: GOTO 1680
1630 P=0
1640 I=10*SGN(A[C,1]-A[A,1])/SQR(1+P*P)
1650 FOR Q=0 TO (A[C,1]-A[A,1])/I
1660 PLOT A[A,1]+Q*I,A[A,2]+Q*I*P: DRAWR 0.4*I,0.4*I*P: NEXT
: RETURN
1670 P=0
1680 I=10*SGN(A[C,2]-A[A,2])/SQR(1+P*P)
1690 FOR Q=0 TO (A[C,2]-A[A,2])/I
1700 PLOT A[A,1]+Q*I*P,A[A,2]+Q*I: DRAWR 0.4*I*P,0.4*I: NEXT
: RETURN
2000 TAG: MOVE 225,380: PRINT Z$;
2010 IF INKEY$="" THEN 2010
2020 TAGOFF: GOTO 20
```

N	C[26,3]	A[26,3,3]		
1	100	HKL	$H\bar{K}L$	$H\bar{L}\bar{K}$
2	001	KLH	$\bar{K}LH$	$\bar{L}\bar{K}H$
3	010	KHL	$\bar{K}HL$	$\bar{L}H\bar{K}$
4	$\bar{1}00$	$\bar{H}KL$	$\bar{H}\bar{K}L$	$\bar{H}\bar{L}\bar{K}$
5	$00\bar{1}$	$KL\bar{H}$	$\bar{K}L\bar{H}$	$\bar{L}\bar{K}\bar{H}$
6	$0\bar{1}0$	$K\bar{H}L$	$\bar{K}\bar{H}L$	$\bar{L}\bar{H}\bar{K}$
7	111	HKL	KLH	LHK
8	$1\bar{1}1$	$H\bar{K}L$	$K\bar{L}H$	$L\bar{H}K$
9	$1\bar{1}\bar{1}$	$H\bar{K}\bar{L}$	$K\bar{L}\bar{H}$	$L\bar{H}\bar{K}$
10	$11\bar{1}$	$HK\bar{L}$	$KL\bar{H}$	$LH\bar{K}$
11	$\bar{1}\bar{1}1$	$\bar{H}\bar{K}L$	$\bar{K}\bar{L}H$	$\bar{L}\bar{H}K$
12	$\bar{1}11$	$\bar{H}KL$	$\bar{K}LH$	$\bar{L}HK$
13	$\bar{1}1\bar{1}$	$\bar{H}K\bar{L}$	$\bar{K}L\bar{H}$	$\bar{L}H\bar{K}$
14	$\bar{1}\bar{1}\bar{1}$	$\bar{H}\bar{K}\bar{L}$	$\bar{K}\bar{L}\bar{H}$	$\bar{L}\bar{H}\bar{K}$
15	110	HKL	KHL	$HK\bar{L}$
16	101	KLH	HLK	$K\bar{L}H$
17	011	LHK	LKH	$\bar{L}HK$
18	$1\bar{1}0$	$H\bar{K}\bar{L}$	$K\bar{H}\bar{L}$	$H\bar{K}L$
19	$10\bar{1}$	$K\bar{L}\bar{H}$	$H\bar{L}\bar{K}$	$KL\bar{H}$
20	011	$L\bar{H}\bar{K}$	$L\bar{K}\bar{H}$	$\bar{L}\bar{H}\bar{K}$
21	$\bar{1}\bar{1}0$	$\bar{H}\bar{K}L$	$\bar{K}\bar{H}L$	$\bar{H}\bar{K}L$
22	$\bar{1}01$	$\bar{K}\bar{L}H$	$\bar{H}\bar{L}K$	$\bar{K}LH$
23	$0\bar{1}1$	$\bar{L}\bar{H}K$	$\bar{L}\bar{K}H$	$L\bar{H}K$
24	$\bar{1}10$	$\bar{H}K\bar{L}$	$\bar{K}H\bar{L}$	$\bar{H}KL$
25	$\bar{1}0\bar{1}$	$\bar{K}L\bar{H}$	$\bar{H}L\bar{K}$	$\bar{K}L\bar{H}$
26	$01\bar{1}$	$\bar{L}H\bar{K}$	$\bar{L}K\bar{H}$	$LH\bar{K}$

Cubic Crystal 4/m3m Class
Control Array and Face Triplet Matrix


440
HKL
H=K?
y
465
L=O?
y
L=O?
y
GOSUB 900
octahedron
455
GOSUB 500
GOSUB 600
Icositetrahedron
475
L=K?
y
475
K=L?
445
GOSUB 700
cube
GOSUB 800
6-faced
octohedron
GOSUB 800
3-faced octahedron
GOSUB 800
Rhombic
Dodecahedron
K=L?
y
GOSUB 800
4-faced cube

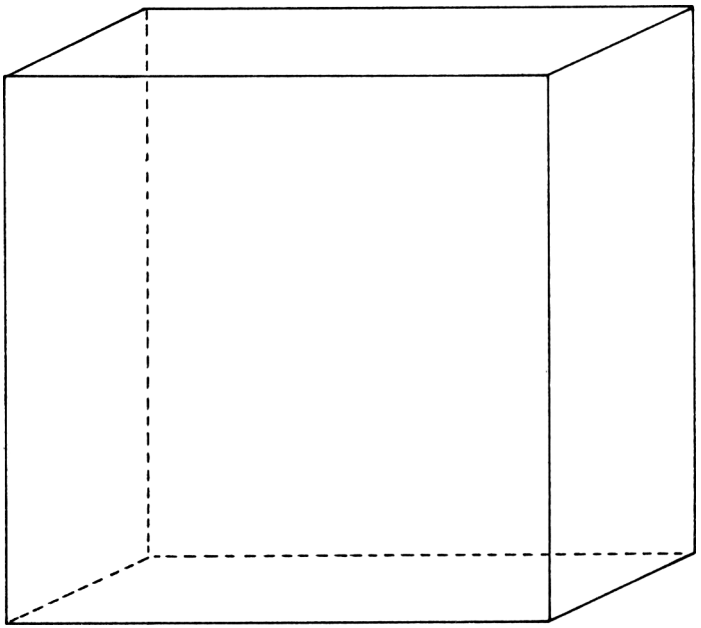


Fig. 3.2 Cubic Crystals
23 and 4/m3m CLASSES
{100} Form Cube

4. CRYSTAL FORM 23 – Ullmannite

The mineral ullmannite belongs to the cubic crystal class with the least symmetry. Whereas the fully symmetrical class has 3 4–fold axes, 4 3–fold axes, 6 2–fold axes and 9 mirror planes as well as a centre, the least symmetrical class only has 4 3–fold axes and 3 2–fold axes. This gives rise to several different external forms based more on the tetrahedron than the bipyramid although some forms e.g. {100} are cubes in both classes.

The program for the 23 form (pronounced two three) is similar to the 4/m3m form (pronounced four over em three em) and many of the subroutines are identical. However, there are some subtle changes and care should be taken if you derive one listing from the other.

The main points of difference are

1. The H, K, and L values used to build up the control array are 111, −111, and 011 and the third column not the first is used. (See lines 20, 30 and 1300).
2. H, K and L are put in ascending order (lines 50 and 60).
3. Lines 100–220 build up the 3×8×6 array in another way but the end result is identical.
4. There are only 20 possible apices in the 23 class so 26 is changed to 20 in lines 230, 250, 300 and 400. A$ to F$ only contain 20 characters.
5. L is used as the scaling factor in the projection formula. Obviously, subroutines 500, 600, 700, 800, and 900 are different as are lines 400–490 which separate the different crystal faces.

```
1 REM Cubic Crystal 23 Class-Ullmannite
2 BORDER 13: INK 0,2: INK 1,0
3 DEFINT B-D,G,H,K,L,N,R-Z: DEF FN A$(A)=MID$(STR$(A),2)
10 DIM C[20,3],X[2]: G=0: H=1: K=1: L=1: GOTO 100
20 ERASE A: G=G+1: IF G=1 THEN H=-1: GOTO 100
30 IF G=2 THEN H=0: GOTO 100
40 PRINT "Next Form"
41 INPUT "h=";H: H=ABS(H)
42 INPUT "k=";K: K=ABS(K)
43 INPUT "l=";L: L=ABS(L): IF H+K+L=0 THEN 40
50 IF H>K THEN SW=K: K=H: H=SW
60 IF K>L THEN SW=L: L=K: K=SW: GOTO 50
70 Z$="CUBIC CRYSTAL FORM "+CHR$(123)+FN A$(H)+FN A$(K)+FN A
$(L)+CHR$(125)
100 DIM A[8,3],D[18],I[3,8,6],P[6,2]
110 X[0]=H: X[1]=K: X[2]=L: N=1
120 FOR A=1 TO 4: A[A,1]=1: A[A+4,1]=-1: NEXT
```

```
130 RESTORE 130: DATA 1,-1,-1,1,-1,1,1,-1
140 FOR A=1 TO 8: READ A[A,2]: NEXT
150 FOR A=1 TO 5 STEP 4: A[A,3]=1: A[A+1,3]=1: A[A+2,3]=-1:
A[A+3,3]=-1: NEXT
160 FOR A=0 TO 6: T=2-(A/2-INT(A/2))*2
170 FOR B=0 TO 2: P[A,B]=X[B]: NEXT
180 SW=X[0]: X[0]=X[T]: X[T]=SW: NEXT
190 FOR A=0 TO 5
200 FOR B=0 TO 2: D[N]=P[A,B]: N=N+1: NEXT: A=A+1
210 FOR B=0 TO 2: D[N]=-P[A,B]: N=N+1: NEXT B,A
220 FOR B=1 TO 8: N=1: FOR C=1 TO 6: FOR A=1 TO 3:.I[A,B,C]=
A[B,A]*D[N]: N=N+1: NEXT A,C,B
230 ERASE A,D,P: DIM A[20,3,3]
240 A$="13573175513771353157"
241 B$="13575713157317531375"
242 C$="13577531153717531375"
243 D$="11111111111133335555"
244 E$="33333331111133335555"
245 F$="55555555555511113333"
250 FOR C=1 TO 20: FOR A=1 TO 3
260 A[C,1,A]=I[A,VAL(MID$(A$,C,1)),VAL(MID$(D$,C,1))]
270 A[C,2,A]=I[A,VAL(MID$(B$,C,1)),VAL(MID$(E$,C,1))]
280 A[C,3,A]=I[A,VAL(MID$(C$,C,1)),VAL(MID$(F$,C,1))]
290 NEXT A,C: ERASE I
291 IF G=0 THEN GOSUB 1200: GOTO 20
292 IF G#1 THEN GOSUB 1210: GOTO 20
293 IF G=2 THEN GOSUB 1220: GOTO 20
300 DIM I[20,3]: FOR A=1 TO 20
310 R=A[A,1,1]: S=A[A,1,2]: T=A[A,1,3]
320 U=A[A.2.1]: V=A[A.2.2]: W=A[A.2.3]
330 X=A[A,3,1]: Y=A[A,3,2]: Z=A[A,3,3]
340 D=R*(V*Z-Y*W)+S*(W*X-Z*U)+T*(U*Y-X*V)
350 IF D=0 THEN 390
360 I[A,1]=(V*Z-Y*W+S*(W-Z)+T*(Y-V))/D
370 I[A,2]=(R*(Z-W)+W*X-Z*U+T*(U-X))/D
380 I[A,3]=(R*(V-Y)+S*(X-U)+U*Y-X*V)/D
390 NEXT: ERASE A
400 DIM A[20,3]: FOR A=1 TO 20
410 A[A,1]=320-58.8*I[A,1]*L++176.4*I[A,2]*L
420 A[A,2]=196-19*I[A,1]-6.5*I[A,2]*L+162*I[A,3]*L: NEXT: ER
ASE I
430 MODE 2
440 IF K=L THEN 480 ELSE IF H=K THEN 460
450 GOSUB 800: GOSUB 900: GOTO 2000
460 IF H=0 THEN GOSUB 500: GOTO 2000
470 GOSUB 600: GOSUB 700: GOTO 2000
480 IF H=K THEN GOSUB 700: GOTO 2000
490 GOSUB 800: GOTO 2000
500 FOR A=1 TO 4: FOR C=5 TO 8
510 IF C[A,1]=-C[C,1] AND C[A,2]=C[C,2] AND C[A,3]=C[C,3] OR
 C[A,1]=C[C,1] AND C[A,2]=-C[C,2] AND C[A,3]=C[C,3] OR C[A,1
]=C[C,1] AND C[A,2]=C[C,2] AND C[A,3]=-C[C,3] THEN GOSUB 530
520 NEXT C,A: RETURN
530 IF C=5 THEN GOSUB 1600: GOTO 550
540 GOSUB 1500
```

```
550 RETURN
600 FOR A=1 TO 4: FOR C=5 TO 8
610 IF C<>A+4 AND C[A,1]=-1 THEN GOSUB 1600: GOTO 630
620 IF C<>A+4 THEN GOSUB 1500
630 NEXT C,A: RETURN
700 FOR A=5 TO 7: FOR C=A+1 TO 8
710 IF C[A,1]=-1 AND C[C,1]=-1 THEN GOSUB 1600: GOTO 730
720 GOSUB 1500
730 NEXT C,A: RETURN
800 FOR A=1 TO 8: FOR C=9 TO 20
810 IF (C[C,1]=C[A,1] OR C[C,1]=0) AND (C[C,2]=C[A,2] OR C[C
,2]=0) AND (C[C,3]=C[A,3] OR C[C,3]=0) THEN GOSUB 830
820 NEXT C,A: RETURN
830 IF A=3 OR A=5 OR A=4 AND H<>0 OR C=19 OR C=20 THEN GOSUB
 1600: GOTO 850
840 GOSUB 1500
850 RETURN
900 FOR A=9 TO 20 STEP 2: C=A+1
910 IF C[A,1]=-1 THEN GOSUB 1600: GOTO 930
920 GOSUB 1500
930 NEXT A: RETURN
1200 FOR A=1 TO 4: GOSUB 1300: NEXT: RETURN
1210 FOR A=5 TO 8: GOSUB 1300: NEXT: RETURN
1220 FOR A=9 TO 20: GOSUB 1300: NEXT: RETURN
1300 FOR B=1 TO 3: C[A,B]=A[A,3,B]: NEXT: RETURN
1500 PLOT A[A,1],A[A,2]: DRAW A[C,1],A[C,2]: RETURN
1600 IF A[A,1]=A[C,1] THEN 1670 ELSE IF A[A,2]=A[C,2] THEN 1
630
1610 P=(A[C,2]-A[A,2])/(A[C,1]-A[A,1])
1620 IF ABS(P)<1 THEN 1640 ELSE P=1/P: GOTO 1680
1630 P=0
1640 I=10*SGN(A[C,1]-A[A,1])/SQR(1+P*P)
1650 FOR Q=0 TO (A[C,1]-A[A,1])/I
1660 PLOT A[A,1]+Q*I,A[A,2]+Q*I*P: DRAWR 0.4*I,0.4*I*P: NEXT
: RETURN
1670 P=0
1680 I=10*SGN(A[C,2]-A[A,2])/SQR(1+P*P)
1690 FOR Q=0 TO (A[C,2]-A[A,2])/I
1700 PLOT A[A,1]+Q*I*P,A[A,2]+Q*I: DRAWR 0.4*I*P,0.4*I: NEXT
: RETURN
2000 TAG: MOVE 225,380: PRINT Z$;
2010 IF INKEY$="" THEN 2010
2020 TAGOFF: GOTO 20
```

N	C[20,3]	A[20,3,3]		
1	111	HKL	KLH	LHK
2	1$\bar{1}\bar{1}$	H$\bar{K}\bar{L}$	K$\bar{L}\bar{H}$	L$\bar{H}\bar{K}$
3	$\bar{1}\bar{1}$1	$\bar{H}\bar{K}$L	$\bar{K}\bar{L}$H	$\bar{L}\bar{H}$K
4	$\bar{1}$1$\bar{1}$	$\bar{H}$K$\bar{L}$	$\bar{K}$L$\bar{H}$	$\bar{L}$H$\bar{K}$
5	$\bar{1}\bar{1}\bar{1}$	H$\bar{K}\bar{L}$	$\bar{K}\bar{L}$H	$\bar{L}$H$\bar{K}$
6	$\bar{1}$11	HKL	$\bar{K}$L$\bar{H}$	$\bar{L}\bar{H}$K
7	11$\bar{1}$	$\bar{H}$K$\bar{L}$	KLH	L$\bar{H}\bar{K}$
8	1$\bar{1}$1	$\bar{H}\bar{K}$L	K$\bar{L}\bar{H}$	LHK
9	101	$\bar{H}\bar{K}$L	HKL	LHK
10	$\bar{1}$01	HKL	$\bar{H}\bar{K}$L	$\bar{L}\bar{H}$K
11	10$\bar{1}$	H$\bar{K}\bar{L}$	$\bar{H}$K$\bar{L}$	L$\bar{H}\bar{K}$
12	$\bar{1}$0$\bar{1}$	$\bar{H}$K$\bar{L}$	H$\bar{K}\bar{L}$	$\bar{L}$H$\bar{K}$
13	011	$\bar{K}$L$\bar{H}$	KLH	HKL
14	01$\bar{1}$	KLH	$\bar{K}$L$\bar{H}$	$\bar{H}$K$\bar{L}$
15	0$\bar{1}$1	K$\bar{L}\bar{H}$	$\bar{K}\bar{L}$H	$\bar{H}\bar{K}$L
16	0$\bar{1}\bar{1}$	$\bar{K}\bar{L}$H	K$\bar{L}\bar{H}$	H$\bar{K}\bar{L}$
17	110	L$\bar{H}\bar{K}$	LHK	KLH
18	1$\bar{1}$0	LHK	L$\bar{H}\bar{K}$	K$\bar{L}\bar{H}$
19	$\bar{1}$10	$\bar{L}\bar{H}$K	$\bar{L}$H$\bar{K}$	$\bar{K}$L$\bar{H}$
20	$\bar{1}\bar{1}$0	$\bar{L}$H$\bar{K}$	$\bar{L}\bar{H}$K	$\bar{K}\bar{L}$H

Cubic Crystal 23 Class
Control Array and Face Triplet Matrix

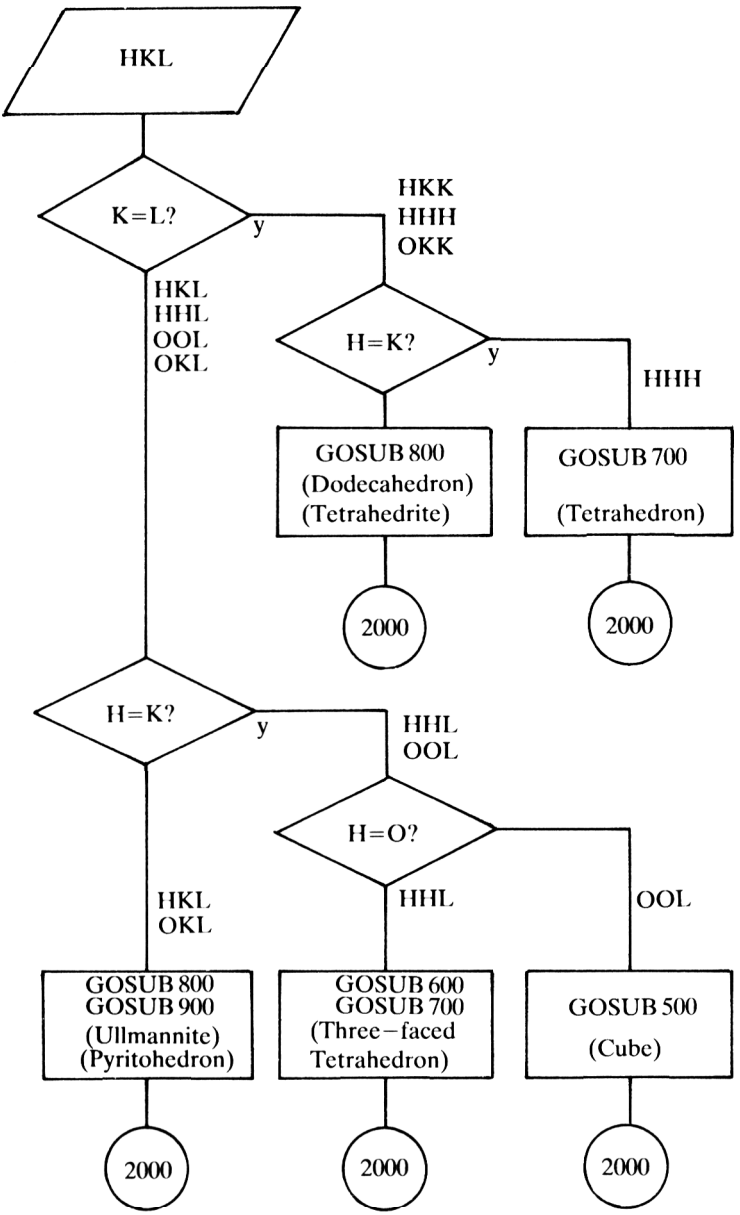


Fig. 4.1 Cubic Crystal 23 Class Face Separation Chart

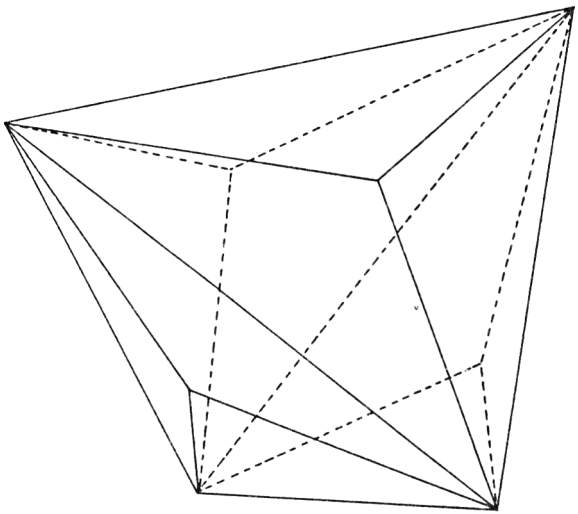


Fig. 4.3 Cubic Crystals
23 CLASS
{211} Form Three faced Tetrahedron

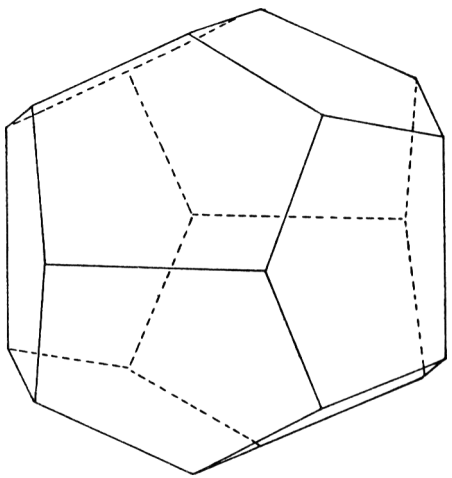


Fig. 4.4 Cubic Crystals
23 CLASS
{210} form Pyritohedron

5. EVALUATION OF A DETERMINANT BY LAPLACE DEVELOPMENT

A determinant is a mathematical shorthand for the sum and difference of a set of products.

$$\begin{vmatrix} a11 & a12 \\ a21 & a22 \end{vmatrix} = a11*a22 - a21*a12$$

and more complicated determinants can be derived by expansion so that

$$\begin{vmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{vmatrix} = a11* \begin{vmatrix} a22 & a23 \\ a32 & a33 \end{vmatrix} - a12* \begin{vmatrix} a21 & a23 \\ a31 & a33 \end{vmatrix} + a13* \begin{vmatrix} a21 & a22 \\ a31 & a32 \end{vmatrix}$$

etc..

As the number of terms is given by factorial N, the size of the printout rapidly increases with N. The different product terms contain all the permutations of the determinant terms and lines 190–290 are the Permute routine. However, the order in which they are generated requires that the sign be changed for every term which is factorial of the odd numbers (i.e. 1,6,120,5040 etc.). Lines 120–160 work out the factorials and lines 250/260 check whether the sign needs changing.

The individual terms and their value are built up in lines 220–240 but the program is a good example of how not to solve a mathematical problem by choosing the 'obvious' method. In terms of execution time the matrix inversion method detailed elsewhere is much more efficient. However, as the routine below only contains products and additions it will always work no matter how ill–conditioned the determinant is.

The defined function in line 40 is for making the print format packed and NL$ is a new line operation.

```
10 REM Evaluation of a determinant by Laplace development
20 DEFINT B,C,M,N,P,Q,S,T,X,Y,Z
30 DEF FN P$(P)=MID$(STR$(P),2): NL$=CHR$(10)+CHR$(13)
40 SYMBOL AFTER 208: H=HIMEM+1: POKE H,0
50 CLS: GOSUB 400:
60 LINE INPUT "Enter the order of the determinant  " ;A$: GO
SUB 340: IF Z=1 THEN Z=0: GOTO 60
70 DIM D[N,N]: PRINT: FOR P=1 TO N: FOR Q=1 TO N
80 PRINT "a"+FN P$(P)+FN P$(Q);"=";: INPUT D[P,Q]: NEXT Q,P
```

```
90 IF N=1 THEN CLS: GOSUB 400: PRINT "The determinant is a11
=";D[1,1]: GOTO 390
100 M=N-1: Y=1+INT(M/2)
110 DIM A[M],B[M],F[N],G[Y]
120 B=3: D$="": D=0: S=0: T=1
130 FOR P=1 TO N: F[P]=P: NEXT P
140 FOR A=1 TO Y: G[A]=2
150 FOR C=3 TO B: G[A]=G[A]*C: NEXT C
160 B=B+2: NEXT A
170 CLS: GOSUB 400
180 PRINT "The determinant is";NL$
190 FOR P=1 TO M: A[P]=1: B[P]=N-P+1: NEXT P
200 FOR P=1 TO M: IF A[P]>B[P] THEN A[P]=1
210 NEXT P: A=1: A$=""
220 FOR P=1 TO N: A=A*D[P,F[P]]: A$=A$+"a"+FN P$(P)+FN P$(F[
P]): NEXT P
230 D=D+T*A: D$=CHR$(44-T)+A$: S=S+1: T=-T
240 PRINT D$;
250 IF N>3 THEN GOSUB 300
260 FOR X=M TO 1 STEP -1
270 Z=F[X]: FOR P=X TO M: F[P]=F[P+1]: NEXT P: F[N]=Z
280 A[X]=A[X]+1: IF A[X]>B[X] THEN NEXT X: GOTO 380
290 GOTO 200
300 FOR P=1 TO Y: IF INT(S/G[P])=S/G[P] THEN T=-T
310 NEXT P: RETURN
320 IF A$="" OR VAL(A$)=0 THEN Z=1
330 RETURN
340 T=1: IF LEFT$(N$,1)="+" THEN T=2
350 FOR P=T TO LEN(A$): IF MID$(A$,T,1)<>"0" AND MID$(A$,P,1
)>="0" AND MID$(A$,P,1)<="9" THEN 370
360 Z=1: RETURN
370 NEXT P: N=VAL(A$): RETURN
380 PRINT: PRINT : PRINT "Det D=";D
390 END
400 LOCATE 6,2: PRINT "DETERMINANT DISPLAY AND VALUE";NL$;NL
$
410 LOCATE 6,3: FOR P=1 TO 29: PRINT CHR$(208);: NEXT
420 PRINT: RETURN
```

EXAMPLE

```
                DETERMINANT DISPLAY AND VALUE

Enter the order of the determinant   4

D(1,1)=? -2
D(1,2)=?  4
D(1,3)=?  7
D(1,4)=?  3
D(2,1)=?  8
D(2,2)=?  2
D(2,3)=? -9
D(2,4)=?  5
D(3,1)=? -4
D(3,2)=?  6
D(3,3)=?  8
D(3,4)=?  4
D(4,1)=?  2
D(4,2)=? -9
D(4,3)=?  3
D(4,4)=?  8

The Determinant is
+a11a22a33a44 -a11a22a34a43 +a11a23a34a42
-a11a23a32a44 +a11a24a32a43 -a11a24a33a42
-a12a23a34a41 +a12a23a31a44 -a12a24a31a43
+a12a24a33a41 -a12a21a33a44 +a12a21a34a43
+a13a24a31a42 -a13a24a32a41 +a13a21a32a44
-a13a21a34a42 +a13a22a34a41 -a13a22a31a44
-a14a21a32a43 +a14a21a33a42 -a14a22a33a41
+a14a22a31a43 -a14a23a31a42 +a14a23a32a41

Det D= 2140
```

6. TRIANGLE

This program uses the geometrical relationships in a triangle to calculate the unknown sides and angles given that you have three known facts including the length of one side. To check that the triangle is in agreement with your data, it will also draw a scaled and labelled drawing if you wish. It was developed to help with engineering drawings.

Lines 30–80 are the sine and cosine rules with FN A and FN C being arccos and arcsin respectively.

IN$ is sliced for the INPUT statements 510, 610, 710 etc..

NL$ is a new line used in PRINT statements.

Lines 190–310 are the MENU

Lines 340–370 and 400–460 give the complete data PRINT presentation. (T=3 represents the ambiguous case where two answers are possible)

SUB'S 500, 600, 700, 800, and 900 are the INPUT routines.
SUB 2000 is the heading and underline
SUB 2040 is for holding
SUB 2070 is a number filter for 1–5 from the MENU
SUB 2100 is "OK to proceed?".
SUB 2120, 2160, 2180, 2200 and 2220 are data checks to ensure that the input data can form a triangle e.g. the sum of the shorter two sides must be greater than the third one etc..
SUB 2400 is the "Invalid Data" statement with FAIL as the carrier. The rest of the program is to draw a scaled triangle. Except for the case when T=3 and VERT>320, the longest side is always drawn as a horizontal line of fixed length, S. Note the use of 0.91 for the screen aspect ratio to get the shape correct.

Lines 2270–2290 find the longest side and lines 2300–2350 then relabel the sides and corners.

Lines 2390–2400 draw the other two sides and if T=3, line 2410 draws the extra line to make two triangles.

The remaining lines 2410–2570 are for labelling the sides and angles at the right place on the screen using the 'TAG PRINT' subroutine represented by SUB 2700.

The subroutine returns to line 490 and then to 180 for the next triangle after 'Press any key to continue'.

```
10 REM Triangle
20 DEG: DIM T$[5]: SYMBOL AFTER 208: HM=HIMEM+1: POKE HM,0
30 DEF FN Z(A,B,C)=(B*B+C*C-A*A)/2/B/C
40 DEF FN A(A)=90-ATN(A/SQR(1-A*A))
50 DEF FN B(A,Y,C)=SQR(A*A+C*C-2*A*C*COS(Y))
60 DEF FN X(A,Y,B)=A*SIN(Y)/B
70 DEF FN C(C)=ATN(C/SQR(1-C*C))
80 DEF FN D(A,Y,X)=A*SIN(Y)/SIN(X)
90 DEF FN U(U$)=-A*(U$="A")-B*(U$="B")-C*(U$="C")
100 DEF FN V(U$)=-X*(U$="X")-Y*(U$="Y")-Z*(U$="Z")
110 IN$="INPUT a, b and c and the angles X, Y and Z (in degr
ees)"
120 NL$=CHR$(10)+CHR$(13): NL$=NL$+NL$: D$=" Degrees"
130 CLS: GOSUB 2000
140 PRINT "  Triangles are defined if three of the six featu
res (three sides plus three    angles) are known provided th
at at leastone is a side."
150 PRINT "  In one case -two sides and one angle- two trian
gles are possible if the angle is not the included one."
160 PRINT "   This program finds the values of the unknown a
ngles and sides, and draws the triangle"
170 GOSUB 2040
180 CLS: GOSUB 2000
190 PRINT "   Choose the combination of known information"
200 T$[1]="   1.  Three Sides"
210 T$[2]="   2.  Two sides and the Included Angle"
220 T$[3]="   3.  Two Sides and the Non-Included          An
gle"
230 T$[4]="   4.  One Side and Two Angles (both          ad
jacent)
240 T$[5]="   5.  One Side and Two Angles (one           ad
jacent)
250 FOR P=1 TO 5: LOCATE 1,3*P+4: PRINT T$[P]: NEXT P
260 LOCATE 1,22: LINE INPUT "Type in 1, 2, 3, 4, or 5  ";T$[
0]
270 GOSUB 2070
280 IF FAIL THEN 180
290 T=VAL(T$[0])
300 ON T GOSUB 500,600,700,800,900
310 S=370
320 IF T=3 THEN 390
330 CLS: GOSUB 2000: LOCATE 1,4
340 PRINT "a=";A;M$;NL$;"b=";B;M$;NL$;"c=";C;M$;NL$;"X=";USI
NG "###.####";X;
350 PRINT D$;NL$;"Y=";USING "###.####";Y;
360 PRINT D$;NL$;"Z=";USING "###.####";Z;
370 PRINT D$
380 C2=0: VERT=0: GOTO 490
390 CLS: GOSUB 2000: LOCATE 1,4
```

```
400 PRINT "a=";A;M$;NL$;"b=";B;M$;NL$;"c1="C1;M$;NL$;"c2=";C
2;M$
410 PRINT: PRINT "X1=";USING "###.####";X1;
420 PRINT D$;NL$;"X2=";USING "###.####";X2;
430 PRINT D$;NL$;"Y=";USING "###.####";Y;
440 PRINT D$;NL$;"Z1="USING "###.####";Z1;
450 PRINT D$;NL$;"Z2="USING "###.####";Z2;
460 PRINT D$
470 C=C1+C2*(SGN(C2)=-1): X=X1: Z=Z1: VERT=A*SIN(Y)*370/C
480 SW=A: A=B: B=SW: SW=X: X=Y: Y=SW: IF VERT>320 THEN S=S*3
20/VERT
490 GOSUB 2040: GOSUB 2260: GOSUB 2040: GOTO 180
500 CLS: GOSUB 2000
510 PRINT: PRINT LEFT$(IN$,16)
520 GOSUB 1000: GOSUB 1010: GOSUB 1020: GOSUB 1060
530 GOSUB 2100: IF FAIL THEN 500
540 H=A: K=B: L=C: GOSUB 2120: IF FAIL THEN 500
550 X=FN A(FN Z(A,B,C)): Y=FN A(FN Z(B,C,A)): Z=FN A(FN Z(C,
A,B))
560 RETURN
600 CLS: GOSUB 2000
610 PRINT: PRINT LEFT$(IN$,8)+MID$(IN$,15,7)+MID$(T$[2],22,1
2)+RIGHT$(T$[2],6)+" Y "+RIGHT$(IN$,12)
620 GOSUB 1000: GOSUB 1020: GOSUB 1040: GOSUB 1060
630 GOSUB 2100: IF FAIL THEN 600
640 GOSUB 2160: IF FAIL THEN 600
650 B=FN B(A,Y,C): X=FN C(FN X(A,Y,B))
660 IF A*COS(Y)>C THEN X=180-X
670 Z=180-X-Y
680 RETURN
700 CLS: GOSUB 2000
710 PRINT: PRINT LEFT$(IN$,10)+MID$(IN$,11,5)+MID$(T$[3],22,
16)+RIGHT$(T$[3],6)+" Y "+RIGHT$(IN$,12)
720 GOSUB 1000: GOSUB 1010: GOSUB 1040: GOSUB 1060
730 GOSUB 2100: IF FAIL THEN 700
740 GOSUB 2180: IF FAIL THEN 700
750 X1=FN C(FN X(A,Y,B)): Z1=180-X1-Y: Z2=X1-Y
760 X2=180-X1: C1=FN D(B,Z1,Y): C2=FN D(B,Z2,Y)
770 RETURN
800 CLS: GOSUB 2000
810 PRINT: PRINT LEFT$(IN$,7)+MID$(IN$,17,16)+MID$(IN$,36)
820 GOSUB 1000: GOSUB 1040: GOSUB 1050: GOSUB 1060
830 GOSUB 2100: IF FAIL THEN 800
840 GOSUB 2200: IF FAIL THEN 800
850 X=180-Y-Z: B=FN D(A,Y,X): C=FN D(A,Z,X)
860 RETURN
900 CLS: GOSUB 2000
910 PRINT: PRINT LEFT$(IN$,7)+MID$(IN$,17,17)+RIGHT$(IN$,19)
920 GOSUB 1000: GOSUB 1030: GOSUB 1050: GOSUB 1060
930 GOSUB 2100: IF FAIL THEN 900
940 GOSUB 2220: IF FAIL THEN 900
950 Y=180-X-Z: B=FN D(A,Y,X): C=FN D(A,Z,X)
960 RETURN
1000 LOCATE 5,8: INPUT "a=";A: RETURN
1010 LOCATE 5,10: INPUT "b=";B: RETURN
```

```
1020 LOCATE 5,12: INPUT "c=";C: RETURN
1030 LOCATE 5,14: INPUT "X=";X: RETURN
1040 LOCATE 5,16: INPUT "Y=";Y: RETURN
1050 LOCATE 5,18: INPUT "Z=";Z: RETURN
1060 LOCATE 5,20: INPUT "Unit of length";M$: RETURN
2000 LOCATE 16,1: PRINT "TRIANGLE"
2010 LOCATE 16,2: FOR P=1 TO 8: PRINT CHR$(208);: NEXT P
2020 RETURN
2040 LOCATE 1,25: PRINT "Press any key to continue"
2050 IF INKEY$="" THEN 2050
2060 RETURN
2070 FAIL=0: IF T$[0]="" THEN FAIL=-1: RETURN
2080 IF T$[0]>="1" AND T$[0]<="5" THEN RETURN
2090 FAIL=-1: RETURN
2100 FAIL=0: LOCATE 1,23: PRINT "      OK? (y/n)  ";: INPUT K
$: K$=UPPER$(K$): IF K$<>"Y" THEN FAIL=-1
2110 RETURN
2120 FAIL=0: IF H<K THEN SW=H: H=K: K=SW
2130 IF K<L THEN SW=K: K=L: L=SW: GOTO 2120
2140 IF H<K+L THEN RETURN
2150 GOSUB 2240: RETURN
2160 FAIL=0: IF Y<180 THEN RETURN
2170 GOSUB 2240: RETURN
2180 FAIL=0: IF B>A*SIN(Y) THEN RETURN
2190 GOSUB 2240: RETURN
2200 FAIL=0: IF Y+Z<180 THEN RETURN
2210 GOSUB 2240: RETURN
2220 FAIL=0: IF X>0 THEN RETURN
2230 GOSUB 2240: RETURN
2240 LOCATE 27,8: PRINT "Invalid Data": GOSUB 2040: FAIL=-1:
 RETURN
2260 CLS: PLOT 135,48: DRAWR S,0: IF VERT>320 THEN P=2: GOTO
 2300
2270 IF A>=B AND A>=C THEN P=0
2280 IF B>=C AND B>=A THEN P=1
2290 IF C>=A AND C>=B THEN P=2
2300 H$=CHR$(65+P): H=FN U(H$)
2310 K$=CHR$(66+P+(P>1)*3): K=FN U(K$)
2320 L$=CHR$(67+P+(P>0)*3): L=FN U(L$)
2330 G$=CHR$(88+P): G=FN V(G$)
2340 I$=CHR$(89+P+(P>1)*3): I=FN V(I$)
2350 J$=CHR$(90+P+(P>0)*3): J=FN V(J$)
2360 SCY=S/H: SCX=SCY*0.91
2370 CJ=SCY*COS(J): CI=SCY*COS(I)
2380 SJ=SCX*SIN(J): SI=SCX*SIN(I)
2390 DRAWR -K*CJ,K*SJ
2400 DRAWR -L*CI,-L*SI
2410 IF T=3 THEN PLOT 135+S-K*CJ,48+K*SJ: DRAWR -K*CJ,-K*SJ
2420 E=260: F=44
2430 TEXT$=H$+"="+LEFT$(STR$(H),7): GOSUB 2700
2440 E=100+(SGN(C2)=-1)*(C2*S/C): F=56+12*(SGN(C2)=-1)
2450 TEXT$=I$+"=": GOSUB 2700
2460 E=80+(SGN(C2)=-1)*(C2*S/C-10): F=40+12*(SGN(C2)=-1)
2470 TEXT$=LEFT$(STR$(I),7): GOSUB 2700
2480 E=510: F=56
```

```
2490 TEXT$=J$+"=": GOSUB 2700
2500 E=500: F=40
2510 TEXT$=LEFT$(STR$(J),7): GOSUB 2700
2520 E=480-K*CJ: F=64+K*SJ
2530 TEXT$=G$+"="+LEFT$(STR$(G),7): GOSUB 2700
2540 E=550-0.5*K*CJ: F=56+0.5*K*SJ
2550 TEXT$=K$+"="+LEFT$(STR$(K),7): GOSUB 2700
2560 E=30+(SGN(C2)=-1)*2*C2*S/C+0.5*L*CI: F=56+0.5*L*SI
2570 TEXT$=L$+"="+LEFT$(STR$(L),7): GOSUB 2700
2580 E=260: F=390: TEXT$="TRIANGLE": GOSUB 2700
2590 PLOT 260,370: DRAWR 128,0
2600 RETURN
2700 FOR Q=1 TO LEN(TEXT$): Z$=MID$(TEXT$,Q,1)
2710 MOVE E,F: TAG
2720 PRINT Z$;
2730 E=E+16+16*(Z$=" "): NEXT Q
2740 TAGOFF: RETURN
```

EXAMPLE

```
TRIANGLE

INPUT a, c and the Included Angle Y(in Degrees)

        a=? 25
        c=? 37
        y=? 45
Unit of Length? mm OK? (y/n)
        a= 25mm
        b= 26.1887849 mm
        c= 37 mm
        X= 42.4549 Degrees
        Y= 45.0000 Degrees
        Z= 92.5451 Degrees
Press any key to continue
```

Fig 6.1 Screen Illustration of TRIANGLE

7. 2–3 TREE STRUCTURE

A 2–3 tree is a tree structure in which every non–zero vertex has either two or three branches. The stored information is carried on the leaves and to find the appropriate leaf each vertex from a leaf to the root carries two additional pieces of information. Reference to the diagram makes this easier to follow. To find a particular leaf you ask two questions at each vertex starting at the root.

1. Is the leaf value less than or equal to the lefthand number?. If so, go down the left branch.
2. If not, is the leaf value less than or equal to the righthand number?. If so, go down the middle branch. Otherwise go down the righthand branch.

To fulfil the 2 or 3 rule, it is easy to show that the maximum number of leaves is 3 ↑ N and the minimum number 2 ↑ N where N is the height of the tree. The following table shows the range of capacities for each size of tree.

Height of tree	*Minimum*	*Maximum*
0	1	1
1	2	3
2	4	9
3	8	27
4	16	81
5	32	243
6	64	729
7	128	2187

Hence with a 2–3 tree of height 7 you can find one leaf out of 2187 with a number of questions which varies from 7 to 13 (weighted average 11) of the type discussed plus 8 questions. "Is this a leaf?" – a total of 19 questions.

With a symmetrical binary tree holding 2047 items the minimum height would be 11 and you would need to ask between 1 and 21 questions (Is it this vertex?. If not, is it less?). (Average also 19.)

The problem which arises with the binary tree is that adding or removing members produces an unsymmetrical tree particularly if the entries happen to be in order (see 'Binary Search Tree') and then the access time becomes significantly different from the best value which is given by a symmetrical tree. The access time for a 2–3 tree is constant and depends only on the

initial tree size.

The flowchart shows the general construction of the program based on a nine choice menu. Following this chart, the program first finds if the program has already been used to establish a data base and if so, asks for it to be INPUT into the computer (SUB 300). If not then a new tree is formed by SUB 5000. This establishes the minimum size of tree required (line 5020) and then, utilising the 'Variable Size Nested Loop', finds the location of the storage positions to fulfil the 2 or 3 rule. The variables B1, B2 and B3 change the end values of the K loop from 0 to 1 so that at each vertex the loop runs from −1 TO +1 (three branches) or −1 TO 0 (two branches) as necessary. The decision tree values are stored in the L and M arrays (left and middle respectively). The last part of the L array (which is bigger than the M array), holds the data to be stored. This is INPUT in line 5150, sorted by 'Heapsort' in SUB 1000 and loaded into L in line 5170.

The program then runs under Menu control (line 200, to which you should return in case of difficulty). Most of the subroutines are straightforward with the exception of 500, 2000 and 3000.

SUB 500 generates the H array which is needed when reorganising the tree in 2000 and 3000. The H array contains the storage positions at spillover i.e. for 3^N+1 values when stored as close as possible within the 2,3 rules. The table gives the first 50 values.

SUB 2000 deals with adding a new value to the existing tree. First of all it finds the value of V for the father (SUB 1800) and three cases are possible

1. The father has 2 sons in which case a third one can be fitted in (SUB 2300).
2. The father already has 3 sons in which case a father with only 2 sons has to be found (loops 2040 and 2070) at position A. The intervening sons are then nudged across to make room for the new entry (lines 2100–2150 for left and lines 2170–2220 for right shift).
3. No father can be found with only 2 sons. In this case the tree has to be reorganised in SUB 2500 to use some of the vacant leaves and branches but still conforming with the 2 or 3 rule. This is done by swapping vacant positions with righthand entries in line 2630 using the function S which is derived from the spillover information in the H array.

After reorganisation, the first two options are tried again to obtain a suitable location. Failure to do so only happens if the tree is full, line 2560

SUB 3000 follows a similar procedure to remove an entry.

The three cases are

1. The father has 3 sons (SUB 3300)
2. The father only has two sons and a father with 3 sons is found and the sons between the two locations are moved right or left (lines 3110–3220)
3. No father has three sons in which case the tree is reorganised (SUB 3500) and cases 1 or 2 will apply unless there are less than 2 ↑ N entries (N is the height of the tree)

Note the use of CHR$(34) for inverted commas in 4430 but WRITE could have been used.

The program can be adapted to store string information or if each piece of information has a unique number this can be stored and used as a pointer to the main string storage array. (See 'Label')

The D,L and M arrays need to be string arrays, X changed to X$ and the various zeros used in comparisons changed to "".

```
9 REM 2-3 Tree Structure
10 DEFINT A,B,F-K,  N,P,Q,V,Z
20 DEF FN L(L)=INT(3^L-0.99999991)
30 DEF FN K(K)=1+K+FN L(K)/2
40 DEF FN S(A,B)=H(A)-H(1+B)+1
50 DEF FN A$(A)=MID$(STR$(A),2)
60 NL$=CHR$(10)+CHR$(13): DIM M$[9]
70 M$[1]="Create a new 2-3 tree"
80 M$[2]="Find a member"
90 M$[3]="Find min/max and number of members"
100 M$[4]="Add a new member"
110 M$[5]="Remove an existing member"
120 M$[6]="Print out a list of members"
130 M$[7]="Save the data on disc"
140 M$[8]="Save the program"
150 M$[9]="Finish"
160 WINDOW #1,1,40,23,25: CLS: GOSUB 1600
170 PRINT "   This program is designed to create a 2-3 tree
storage structure  using a disc unit for the storage and ret
rieval of  data."
180 PRINT "   If you have already created the tree and store
d it then press"+CHR$(34)+"Y  and <ENTER>"+CHR$(34)+"else ju
st "+CHR$(34)+"<ENTER>"+CHR$(34)
190 LINE INPUT K$: K$=UPPER$(K$): IF K$="Y" THEN GOSUB 300 E
LSE GOSUB 5000: GOSUB 500
200 CLS: GOSUB 1600
210 FOR P=1 TO 9: LOCATE 2,2+2*P: PRINT P;CHR$(8);". ";M$[P]
: NEXT
220 LOCATE #1,1,1: PRINT #1,"Type in your choice 1-9";
230 LOCATE 25,23: INPUT M$[0]: GOSUB 1650
```

```
240 IF FAIL THEN 200
250 K=VAL(M$[0])
260 ON K GOSUB 4700,4500,4000,2000,3000,4600,4100,4300,4400
270 GOTO 200
299 REM Load the data from disc
300 CLS: GOSUB 1600: PRINT "Is the disc unit ready to INPUT
the stored data (Y/N)?";NL$
310 IF INKEY$<>"Y" AND INKEY$<>"y" THEN 310
320 INPUT "Name of data file";TREE$
330 OPENIN TREE$
340 INPUT #9,N,M: AL=FN L(N+1)/2: AM=FN L(N)/2: DIM L[AL],M[
AM]
350 FOR P=1 TO AL: INPUT #9,L[P]: NEXT: INPUT #9,SMALL
360 FOR P=1 TO AM: INPUT #9,M[P]: NEXT: INPUT #9,LARGE
370 CLOSEIN
380 GOSUB 500
390 RETURN
499 REM Deriving the H array
500 NH=FN K(N): DIM H[NH]: H[1]=1
510 I=0: J=0: K=1: Y=1: Z=-1
520 FOR P=1 TO FN K(N-1)
530 IF H[K]=9 THEN I=1: Y=7: J=0: Z=1
540 FOR Q=-1 TO -(I>0)
550 IF H[P]<>0 THEN H[K+1]=3*H[P]+Q: K=K+1: J=J+1
560 NEXT Q
570 IF J<3*FN L(Z) THEN 600
580 I=0
590 NZ=FN L(Z): IF P>3 THEN IF H[Y+3*NZ+4]/H[Y]=3 THEN I=1:
J=0: Y=Y+3*NZ+4: Z=Z+1
600 NEXT P
610 RETURN
999 REM Heapsort
1000 FOR P=INT(M/2) TO 1 STEP -1: R=P
1010 S=R+R: T=S+1
1020 IF D[R]<D[S] THEN GOSUB 1170: GOTO 1050
1030 IF T<=M THEN IF D[R]<D[T] THEN GOSUB 1200: GOTO 1050
1040 GOTO 1060
1050 IF R<=M/2 THEN 1010
1060 NEXT P
1070 FOR P=M TO 3 STEP -1: SW=D[1]: D[1]=D[P]: D[P]=SW: R=1
1080 S=R+R: T=S+1
1090 IF D[R]<D[S] THEN GOSUB 1150: GOTO 1120
1100 IF T<P THEN IF D[R]<D[T] THEN GOSUB 1200: GOTO 1120
1110 GOTO 1130
1120 IF R<=(P-1)/2 THEN 1080
1130 NEXT P
1140 SW=D[1]: D[1]=D[2]: D[2]=SW: RETURN
1150 IF T>=P THEN 1190
1160 GOTO 1180
1170 IF T>M THEN 1190
1180 IF D[S]<D[T] THEN 1200
1190 SW=D[R]: D[R]=D[S]: D[S]=SW: R=R+R: RETURN
1200 SW=D[R]: D[R]=D[T]: D[T]=SW: R=R+R+1: RETURN
1499 REM Number
1500 FOR H=N-1 TO 0 STEP -1
```

```
1510 FOR V=FN L(H)/2+1 TO FN L(H+1)/2
1520 T=H: VL=3*V-1: VM=VL+1
1530 IF T=N-1 THEN 1570
1540 VL=3*VL-(L[3*VL+1]>0)
1550 VM=3*VM-(L[3*VM+1]>0)
1560 T=T+1: GOTO 1530
1570 L[V]=L[VL]: M[V]=L[VM]
1580 NEXT V,H
1590 RETURN
1599 REM Title
1600 LOCATE 11,2: PRINT "2-3 TREE STRUCTURE": LOCATE 11,3
1610 FOR P=1 TO 18: PRINT CHR$(208);: NEXT: PRINT
1620 RETURN
1649 REM M$[0] test
1650 FAIL=0: IF M$[0]="" THEN 1670
1660 IF LEN(M$[0])=1 AND M$[0]>="1" AND M$[0]<="9" THEN RETU
RN
1670 FAIL=-1: RETURN
1699 REM Tally of members
1700 M=M+1: SMALL=0
1710 IF M=3^N THEN PRINT "Tree structure full": LARGE =-1: G
OSUB 4750
1720 RETURN
1730 M=M-1: LARGE=0
1740 IF M=2^N THEN PRINT "Tree at minimum size": SMALL=-1: G
OSUB 4750
1750 RETURN
1799 REM Member check
1800 V=1: H=1: MEMBER=0
1810 IF H<N THEN GOSUB 1850: H=H+1: GOTO 1810
1820 V3=3*V
1830 IF X=L[V3-1] OR X=L[V3] OR X=L[V3+1] THEN MEMBER=-1
1840 RETURN
1850 IF X<=L[V] THEN V=3*V-1: RETURN
1860 IF X<=M[V] THEN V=3*V: RETURN
1870 V=3*V+1: RETURN
1999 REM Addson
2000 CLS: GOSUB 1600: INPUT "New Member";X: IF X<=0 THEN 200
0
2005 IF LARGE=-1 THEN PRINT "Tree Full": GOSUB 4750: RETURN
2010 GOSUB 1800: IF MEMBER THEN PRINT "Already a member": GO
SUB 4750: RETURN
2020 IF L[V]<>0 AND L[3*V+1]=0 THEN GOSUB 2300: GOSUB 1500:
GOTO 1700
2030 GOSUB 2350
2040 FOR P=FN L(N-1)/2+1 TO V-1
2050 IF L[P]<>0 THEN IF L[3*P+1]=0 THEN A=P: GOTO 2100
2060 NEXT P
2070 Z=Z+1: IF V+1>FN L(N)/2 THEN 2500 ELSE FOR P=V+1 TO FN
L(N)/2
2080 IF L[P]<>0 THEN IF L[3*P+1]=0 THEN A=P: GOTO 2170
2090 NEXT P: GOTO 2500
2100 A=3*A+1: IF A=Z THEN L[A]=X: GOTO 2230
2110 B=A+1
2120 IF B<Z THEN IF L[B]=0 THEN B=B+1: GOTO 2120
```

```
2130 L[A]=L[B]: IF L[B]<>0 THEN A=B
2140 IF B<Z THEN GOTO 2110
2150 L[A]=X
2160 GOSUB 1500: GOTO 1700
2170 A=3*A+1
2180 B=A-1
2190 IF B>Z THEN IF L[B]=0 THEN B=B-1: GOTO 2190
2200 L[A]=L[B]: IF B>=Z THEN A=B
2210 IF B>Z THEN 2180
2220 L[B]=X
2230 GOSUB 1500: GOTO 1700
2299 REM Case where father has only two sons
2300 V3=V*3
2310 IF X<=L[V] THEN L[V3+1]=L[V3]: L[V3]=L[V3-1]: L[V3-1]=X
: RETURN
2320 IF X<=M[V] THEN L[V3+1]=L[V3]: L[V3]=X: RETURN
2330 L[V3+1]=X: RETURN
2349 REM Finding Z
2350 V3=V*3: IF L[V]=0 THEN Z=V3-1: RETURN
2360 IF X<L[V3-1] THEN Z=V3-2: RETURN
2370 IF X<L[V3] THEN Z=V3-1: RETURN
2380 IF X<L[V3+1] THEN Z=V3: RETURN
2390 Z=V3+1: RETURN
2499 REM Reorganising the tree
2500 FOR H=N-2 TO 0 STEP -1
2510 FOR P=FN L(H)/2+1 TO FN L(H+1)/2
2520 V=P
2530 IF L[V]=0 THEN 2550
2540 IF L[3*V+1]=0 THEN GOSUB 2600: GOSUB 1500: GOSUB 1800:
GOTO 2020
2550 NEXT P
2560 NEXT H: PRINT "Adding this member would make the tree o
ut of memory": GOSUB 4750: LARGE=-1: RETURN
2600 A=N-H-1: A1=2*(FN L(A)+1): SA=FN K(A)
2610 IF H<N THEN V=3*V+1: H=H+1: GOTO 2610
2620 V1=V-A1: V1K=V1-SA-1: FOR P=FN K(A+1) TO FN K(A)+1 STEP
 -1: V1S=V1+FN S(P,SA):
2630 SW=L[V1S]: L[V1S]=L[V1K+P]: L[V1K+P]=SW
2640 NEXT P: RETURN
2999 REM Removeson
3000 CLS: GOSUB 1600: INPUT "Which entry would you like to r
emove";X: IF X<=0 THEN 3000
3010 IF SMALL=-1 THEN PRINT "Tree at minimum size": GOSUB 47
50: RETURN
3020 GOSUB 1800: IF NOT MEMBER THEN PRINT "Not a member": GO
SUB 4750: RETURN
3030 IF L[3*V+1]<>0 THEN GOSUB 3300: GOSUB 1500: GOTO 1730
3040 GOSUB 3450
3050 FOR P=FN L(N-1)/2+1 TO V-1
3060 IF L[P]<>0 AND L[3*P+1]<>0 THEN A=P: GOTO 3110
3070 NEXT P
3080 IF V+1>FN L(N)/2 THEN 3500 ELSE FOR P=V+1 TO FN L(N)/2
3090 IF L[P]<>0 THEN IF L[3*P+1]<>0 THEN A=P: GOTO 3170
3100 NEXT P: GOTO 3500
3110 A=3*A+1
```

```
3120 B=Z-1
3130 IF B>A THEN IF L[B]=0 THEN B=B-1: GOTO 3130
3140 L[Z]=L[B]: Z=B
3150 IF B>A THEN 3120
3160 L[B]=0: GOSUB 1500: GOTO 1730
3170 A=3*A+1
3180 B=Z+1
3190 IF B<A THEN IF L[B]=0 THEN B=B+1: GOTO 3190
3200 L[Z]=L[B]: Z=B
3210 IF B<A THEN 3180
3220 L[B]=0: GOSUB 1500: GOTO 1730
3299 REM Case where father has three sons
3300 IF X=L[V] THEN L[V3-1]=L[V3]: L[V3]=L[V3+1]: L[V3+1]=0:
 RETURN
3310 IF X=M[V] THEN L[V3]=L[V3+1]: L[V3+1]=0: RETURN
3320 L[V3+1]=0: RETURN
3449 REM Finding Z
3450 IF X=L[V] THEN Z=V3-1: RETURN
3460 Z=V3: RETURN
3499 REM Reorganising the tree
3500 FOR H=N-2 TO 0 STEP -1
3510 FOR P=FN L(H)/2+1 TO FN L(H+1)/2
3520 V=P
3530 IF L[V]=0 THEN 3550
3540 IF L[3*V+1]<>0 THEN GOSUB 3570: GOSUB 1500: GOSUB 1800:
 GOTO 3030
3550 NEXT P,H
3560 PRINT "Removing this member would leave too few members
 for this size of tree": GOSUB 4750: SMALL=-1: RETURN
3570 A=N-H-1: A1=2*(FN L(A)+1): SA=FN K(A)
3580 IF H<N THEN V=3*V+1: H=H+1: GOTO 3580
3590 V1=V-A1: V1K=V1-SA-1: FOR P=FN K(A)+1 TO FN K(A+1): V1S
=V1+FN S(P,SA)
3600 SW=L[V1S]: L[V1S]=L[V1K+P): L[V1K+P]=SW
3610 NEXT P
3620 RETURN
3999 REM Number of members
4000 CLS: GOSUB 1600
4010 PRINT "Number of members is";M;NL$
4020 SM=FN L(N): PRINT "Smallest member is";L[SM/2+1];NL$
4030 V=1: H=0
4040 WHILE H<N: V=3*V-(L[3*V+1]<>0): H=H+1: WEND
4050 PRINT "Largest member is";L(V)
4060 GOSUB 4750: RETURN
4099 REM Save the data
4100 CLS: GOSUB 1600
4110 PRINT "Is the disc unit ready to record the data (Y/N)?
";NL$
4120 IF INKEY$<>"Y" AND INKEY$<>"y" THEN 4120
4130 INPUT "Name of data file";TREE$
4140 OPENOUT TREE$
4150 PRINT #9,N,M
4160 FOR P=1 TO FN L(N+1)/2: PRINT #9,L[P]: NEXT: PRINT #9,S
MALL
```

```
4170 FOR P=1 TO FN L(N)/2: PRINT #9,M[P]: NEXT: PRINT #9,LAR
GE
4180 CLOSEOUT: PRINT "OK": GOSUB 4750
4190 RETURN
4299 REM Save program
4300 CLS: GOSUB 1600
4310 PRINT "Is the disc unit ready to record the program (Y/
N)?";NL$
4320 IF INKEY$<>"Y" AND INKEY$<>"y" THEN 4320
4330 SAVE "23TREE"
4340 PRINT "The program saved on disc is now included in the
 index."
4370 CAT
4380 GOSUB 4750
4390 RETURN
4399 REM Finish
4400 CLS: GOSUB 1600
4410 PRINT "Do you really want to finish?";NL$
4420 PRINT "Have you recorded the data?";NL$
4430 PRINT "Press ";CHR$(34);"Y";CHR$(34);" for finish else
<ENTER>";: INPUT K$
4440 K$=UPPER$(K$): IF K$="Y" THEN NEW ELSE RETURN
4499 REM Member?
4500 CLS: GOSUB 1600
4510 INPUT "Which member do you wish to check";X
4520 GOSUB 1800
4530 IF MEMBER THEN PRINT NL$;X;" is a member" ELSE PRINT NL
$;X;" is not a member"
4540 GOSUB 4750: RETURN
4599 REM List of members
4600 CLS: GOSUB 1600
4610 PRINT "The following is a list of members": PRINT
4620 FOR P=FN L(N)/2+1 TO FN L(N+1)/2
4630 PRINT L[P];
4640 NEXT P: PRINT
4650 GOSUB 4750: RETURN
4699 REM Erase arrays
4700 ERASE H,L,M
4710 GOSUB 5000: GOSUB 500
4720 RETURN
4749 REM Hold
4750 PRINT #1,"Press any key to continue"
4760 IF INKEY$="" THEN 4760
4770 RETURN
4999 REM New 2-3 Tree
5000 CLS: GOSUB 1600: INPUT "Number of values to be stored";
M: IF M<2 THEN 5000
5010 Y=M+1: N=-1
5020 WHILE Y>1: Y=Y/2: N=N+1: WEND
5030 V=1: Z=1: F=2^N: G=3*2^(N-1): NV=FN L(N+1)/2: NM=FN L(N
)/2
```

The above program is written specifically for the CPC-6128. Owners of the CPC-464 will need to replace lines 299-390, 4099-4180 and 4299-4380 with tape-handling commands.

The H Array

N	H[N]	N	H[N]	N	H[N]
1	1	18	41	35	58
2	2	19	42	36	68
3	3	20	43	37	69
4	5	21	44	38	70
5	6	22	45	39	71
6	8	23	46	40	72
7	9	24	47	41	73
8	14	25	48	42	77
9	15	26	49	43	78
10	16	27	50	44	80
11	17	28	51	45	81
12	18	29	52	46	122
13	19	30	53	47	123
14	23	31	54	48	124
15	24	32	55	49	125
16	26	33	56	50	126
17	27	34	57		

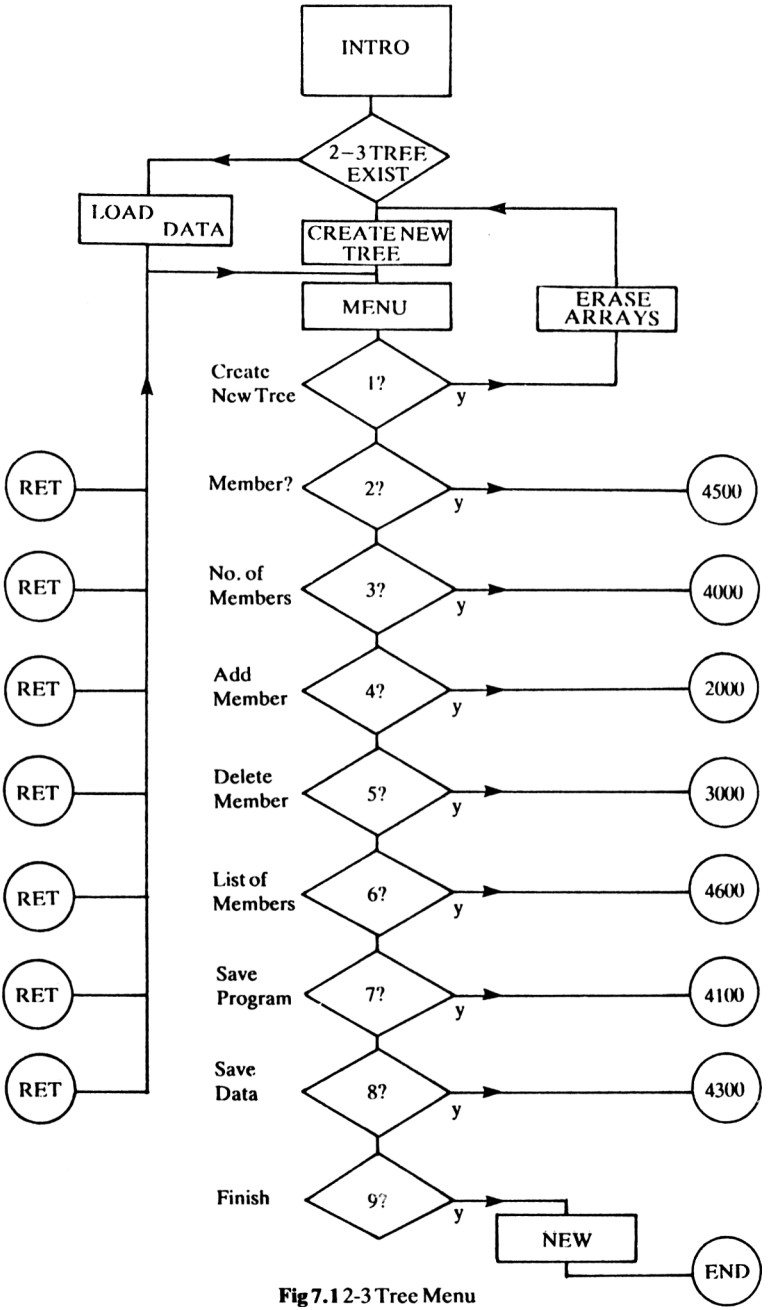
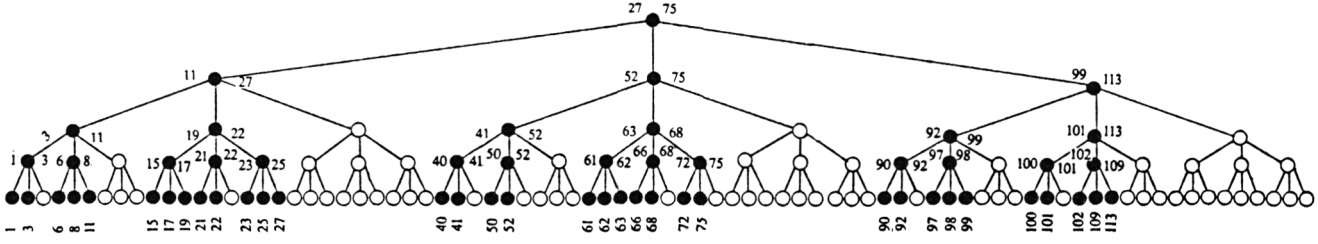


Fig 7.1 2-3 Tree Menu

Fig 7.2 2–3 Tree Structure



Note that the lefthand number is the largest entry down the immediately lefthand path and the righthand number is the largest entry down the immediately centre path.

INDEX

The index is compiled to refer to particular programs or subroutines which illustrate the various uses of statements, functions and cyphers and of course is not exhaustive.

As the subroutines are in alphabetical order, the numbers following the title are the line numbers in the routine unless indicated otherwise (e.g. Introduction, explanation, SUB). Reference just to the subroutine or routine titles are general in nature.

Problems to Solve? Programs to Write?

This unique book is a collection of over 100 Amstrad subroutines to solve your programming problems. You simply add the subroutines required to your own programs – saving days of extra programming, and solving so many tedious problems.

The coverage is vast, and includes:

- checking of input data
- sorting
- data storage and retrieval
- graphics image manipulation
- statistical analysis
- advanced mathematical techniques, including matrix manipulation.

Towards the end of the book, a number of complete programs are included to illustrate the effective use of these subroutines.

All subroutines are written in BASIC for the Amstrad 464/664/6128 machines. Where appropriate, comments are made on the use of tape or disk storage..

About Sigma
We publish a wide range of books for programmers, scientists and technologists – and we welcome your proposals for new books.

Sigma Press
98a Water Lane
Wilmslow
Cheshire
SK9 5BB

GB £ NET +006.95

ISBN 1-85058-046-4

00695

9 781850 580461

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/