A REWARD BOOK

$9.95

THE ZX 81 POCKET BOOK

TREVOR TOMS

THE ZX 81 POCKET BOOK

THE ZX 81 POCKET BOOK

TREVOR TOMS

RESTON PUBLISHING COMPANY, INC.
Reston, Virginia
A Prentice-Hall Company

ISBN 0-8359-9525-9
     0-8359-9524-0 (pbk.)

This book was created using WordStar on a Panasonic JD800 with Diablo 630 printer and Courier 10 typeface.

The U.S. Edition of THE ZX81 POCKET BOOK is produced by Reston Publishing Company, Inc., A Prentice-Hall Company, Reston, Virginia, under license of Phipps Associates, 99 East Street, Epsom, Surrey, England KT17 1EA.

Sales restricted to the United States, its territories and possessions, and Canada.

10 9 8 7 6 5 4 3 2

Printed in the United States of America.

TABLE OF CONTENTS

Graphics by:
Chiltern Creative Services
Suffolk House,
313 London Road,
High Wycombe, Bucks
Tel: High Wycombe 35353

INTRODUCTION

1.1 About This Book

The ZX81 has followed rapidly on the heels of the ZX80, which has outsold almost every other type of personal computer made. The main reasons for its success are its low cost and simplicity - a unique combination.

In March 1981, Sinclair released the new ZX81 - more features, better casing, and even lower cost! How can it possibly fail to be another winner? Most of the computer press has given glowing reports of its capabilities. If you are learning, then it really is ideal.

This is our second book along these lines (the first is The ZX80 Pocket Book) and the content is almost entirely new. I have attempted to keep a similar quality to this book, although the style is hopefully less clinical (we all learn as we go along...) and the layout less fixed.

If you have a ZX80, you should purchase an 8K ROM upgrade before launching too deeply into this book, as all the programs have been written for the ZX81, and not for the ZX80 with "And ZX81!" just printed on the cover. This is important as the two machines are really quite different.

The vast majority of ZX81's sold will not have the additional 16K RAM pack fitted and so there is quite a large selection of assorted "goodies" for these people. Personally, I feel that the machine just cannot be used to its best without this extra RAM pack and I would strongly urge you to buy one if the computing "bug" gets hold of you in any way. Perhaps Santa may be feeling a bit more extravagant - who knows?

Newcomers to computing will hopefully find something more challenging than just another set of games to copy and run - I sincerely hope that you will learn along the way and maybe have a laugh or two. If you <u>are</u> relatively new to programming, you will find it very easy to fall into the habit of purely typing in programs written by other people and running them. This is <u>not</u> programming - most people can type. Try to develop a style of your own, using some of the ideas in this book to fuel your imagination. There is much more pleasure to be gained from running a program that is your own handiwork - especially if your friends become rather attached to it.

I have also tried to expand on the "useful subroutines", as I think these give a much clearer picture of how certain features can be highlighted and how you can stretch the ZX81 to its utmost. Again, newcomers (isn't there a better word?) will find something to learn from these as they can be incorporated into new programs very easily.

In case you're lazy and can't be bothered to type some of the larger programs, a cassette tape is available which contains all the programs in this book. Write (or phone) to the address (or number) given at the front of the book for details of price etc.

One last point - if you find a mistake in a program, I would be only too pleased to know about it, but bitter experience tells that most errors are simple typing mistakes that can go unnoticed for quite a while. These mistakes (whether yours or mine) can tend to detract from your enjoyment of the program when you've spent a considerable time typing it all in. Don't despair! By obtaining a copy of the tape, you will find, I hope, that any problem will vanish - it is certainly much easier for me to fix a problem in the tapes than it is in the books - and I will endeavour to ensure that the tapes are always up-to-date with anything found (plus a couple of extra "goodies" into the bargain!).

Wherever a program is contained on the tape (some of the extremely small examples are not on the tape), you'll see something like:-

Tape name: "ALZAN"

....written alongside the program listing.

1.2 The Structure of this Book

I have used several of my own conventions throughout this book, and this seems an appropriate point to mention them:-

Program listings

1) All keyword commands are underlined.

2) All keyword functions and single keystroke "tokens" (such as **<>** or **>=** etc) are printed in bold face.

3) Printed text which spreads onto more than one line does not necessarily appear in exactly the same way that it will appear on your screen when you type it. This is for clarity of reading. An example (taken from the "Ski Run" program) is:-

```
9060 PRINT " YOU MAY INITIALLY SELECT THE",
           "DIFFICULTY OF COURSE YOU WISH",
           "TO ATTEMPT."
```

This has been laid out so that the lines of text appear in columns as they will when the program is run. Notice that the first two lines end with a comma, indicating that there is more to follow. This comma must be typed, and the next line typed immediately following, so that as you enter them, you will see:-

```
9060 PRINT " YOU MAY INITIALLY S
ELECT THE","DIFFICULTY OF COURSE
 YOU WISH","TO ATTEMPT."
```

4) The listings are annotated with my comments to assist you in studying my programming methods. These are placed on the right-hand side of the page and enclosed by a bracket - e.g.

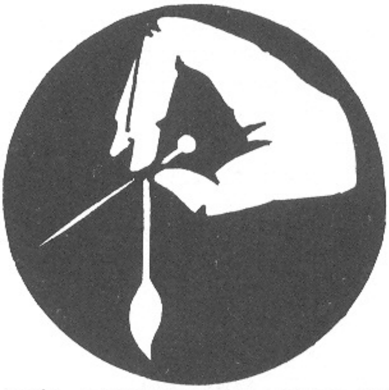
```
200 LET X=1                     (this is a comment
```

Text Passages

The text contains many small working examples of programs and subroutines for you to run. Some of these programs are designed to highlight a particular feature, especially in the section "Efficient Programming", where I have included many "benchmark" programs. These should be timed so that you can see the various effects that different methods of programming have on the resulting program. A set of results is included at the end of the book.

Pinning the Tail on the Donkey

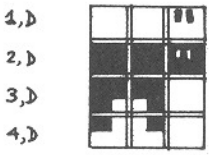
Suitable for : 1K RAM



Here's a piece of fun to start with - try to pin the tail on the donkey. The game is run in fast mode so that you cannot see the donkey all the time. As soon as you press a key, the screen blanks and you have to guess how long to keep pressing the key. When you take your finger off, the tail will stretch across to the donkey. If you succeed in touching the donkey, then your score is given. If you wait too long, though, you may well cause the donkey some damage!

Try to pin the tail on in the minimum number of "peeks".

The donkey is drawn in line 40, and should look like (in larger-size characters):-



Tape name: "DONKEY"

```
 1 REM DONKEY
 2 REM
10 FAST
20 LET T=10                                  (memory saver
30 LET D=INT (RND*T)+19                      (position donkey
40 PRINT AT 1,D;"  ▘▘";AT 2,D;"███";AT 3,D;"▛▛";AT 4,D;"▖▖"
                                             (draw donkey
  50 LET N=1                                  (no. of turns
  60 LET X=NOT N                              (starting position
 100 PAUSE 4E4                                (display
 110 POKE 16437,255
 120 IF INKEY$<>"" THEN GOSUB 1000            (check for key
 130 IF C>=D THEN GOTO 300                    (gone past donkey?
 140 IF X=D*2-1 THEN GOTO 200                 (pinned on properly?
 150 LET N=N+1                                (update no. of turns
 160 GOTO 100                                 (have another go
 200 PRINT AT T,T;"PERFECT";AT 14,0;N;" TURNS"  (win!
 210 GOTO 400
 300 IF X<=63 THEN PRINT AT T,T;"OUCH."       (lose.
 400 INPUT Y$                                 (wait for another go
 410 CLS
 420 RUN
1000 PLOT X,39                                (draw the tail
1010 LET C=INT (X/2)                          (see if hit donkey
1020 IF C-D>=0 AND C-D<3 THEN UNPLOT X,39     (...if so, unplot.
1030 IF INKEY$="" THEN RETURN                 (return if no key
1040 LET X=X+1                                (next position
1050 IF X<64 THEN GOTO 1000                   (still on screen?
1060 PRINT AT T,T;"OH DEAR."                  (gone off screen
1070 RETURN
```

SEARCHING FOR STRINGS

2.1 Why search for strings?

Perhaps the use of a string-searching subroutine can best be illustrated by an example. Imagine a program along these lines:-

```
100 PRINT "WHAT IS A BABY HARE CALLED?"
110 INPUT A$
```

Now what? When the program is run, any answer at all could be given which contains the right word - "leveret". But the string A$ might have been typed as:-

```
A LEVERET
LEVERET
IT IS CALLED A LEVERET
```

...and so on.

This is where the subroutine comes in. It allows you to search for the existence of one string inside another, and to be able to see if the string is not there. It is roughly equivalent to the MicroSoft Basic INSTR function (for those of you who know MicroSoft Basic).

Two variables are required by the subroutine:-

A$ which contains the string to be scanned
W$ which contains the string to look for within A$

On returning, variable X will contain either zero, in which case A$ did not contain the string W$, or it will contain the address of the first character of the first occurrence of W$ within A$. Here's the routine:-

```
9200 REM X=INSTR(A$,W$)              (Tape name: "INSTR"
9210 IF LEN W$=0 THEN GOTO 9290      (to stop error 3
9220 LET X=1                         (initial character no.
9230 LET Y=LEN W$                    (end character number
9240 IF Y>LEN A$ THEN GOTO 9290      (have we gone beyond end?
9250 IF W$=A$(X TO Y) THEN RETURN    (is it in W$?
9260 LET X=X+1                       (otherwise next character
9270 LET Y=Y+1                       (also update end marker
9280 GOTO 9240                       (try again
9290 LET X=0                         (set to zero if not found
9295 RETURN                          (end subroutine
```

Let's see that first example program again, but this time using our new "in-string" subroutine:-

```
100 REM SILLY QUIZ GAME              (Tape name: "QUIZ" - 16K
110 LET SCORE=0
115 LET TOTAL=0
120 LET Q$="WHAT IS A BABY HARE CALLED?"
130 LET W$="LEVERET"
140 GOSUB 8000
150 LET Q$="WHAT IS THE CAPITAL OF CANADA?"
160 LET W$="OTTAWA"
170 GOSUB 8000
180 LET Q$="HOW MANY DAYS ARE IN JULY?"
190 LET W$="31"
200 GOSUB 8000
210 LET Q$="WHAT IS 54.2 * 29.333?"
220 LET W$=STR$ (54.2*29.333)
230 GOSUB 8000
240 LET Q$="WHO IS THE GREATEST?"
250 LET W$="I AM"
260 GOSUB 8000
270 LET Q$="WHEN WAS THE GREAT PLAGUE?"
280 LET W$="1665"
290 GOSUB 8000
300 LET Q$="WHO IS ""THE IRON LADY""?"
310 LET W$="THATCHER"
320 GOSUB 8000
```

```
 330 LET Q$="WHAT IS YOUR SCORE SO FAR?"
 340 LET W$=STR$ SCORE
 350 GOSUB 8000
 400 SCROLL
 410 SCROLL
 420 PRINT "YOU SCORED ";SCORE;" OUT OF ";TOTAL
 430 GOTO 9999
8000 SCROLL
8005 SCROLL
8010 PRINT Q$
8020 INPUT A$
8025 LET TOTAL=TOTAL+1
8030 SCROLL
8040 PRINT A$
8050 GOSUB 9200
8055 SCROLL
8060 IF X THEN GOTO 8110
8070 PRINT "THAT IS WRONG"
8080 SCROLL
8090 PRINT "THE ANSWER IS """;W$;""""
8100 RETURN
8110 PRINT """";W$;""" IS CORRECT"
8120 LET SCORE=SCORE+1
8130 RETURN
9999 STOP
```

Variable Q$ holds the question, W$ holds the answer, and subroutine 8000 asks the question, inputs an answer and checks to see if it is valid. As long as the answer entered contains the keyword in W$, the score is updated. So it makes the program much more fun to use, because there is no restriction on typing just a single word or number. You type in what you want, and the subroutine sorts it all out for you.

A couple of those questions are trick ones, but if you read through the rest of this book, you'll find a way to ensure that every answer you give to this quiz is correct - even if you haven't seen the question. How's it done? I'll leave the answer at the end of the book if you haven't found out by then...

Ski Run

Suitable for : 16K RAM

If you always wanted to try the slalom runs in the winter olympics, then here's your big chance.

You must guide yourself down the slope avoiding the posts on the way. Use the "5" and "8" keys to direct yourself left and right accordingly.

The program as shown here contains two different courses which are chosen at random. After the program listing, you will find instructions on how you may alter courses or enter some of your own.

```
   1 REM *** SKI RUN ***                  (Tape name: "SKI"
   2 REM
   3 REM
  10 GOTO 100
  20 REM CHECK FOR MOVEMENT
  30 LET D=CODE INKEY$                    (get key code
  40 IF NOT D THEN RETURN                 (no valid key pressed
  50 IF D=33 THEN LET S=S-1               (alter skier position
  60 IF D=36 THEN LET S=S+1
  70 LET S=S-INT (S/32)*32                (ensure skier on screen
  80 RETURN
 100 REM INITIALISE GAME
 110 GOSUB 9000                           (print rules
 120 GOSUB 8000                           (select course
 130 LET P=VAL C$(1 TO 2)                 (course start position
 140 LET S=VAL C$(3 TO 4)                 (skier start position
 150 LET H=0                              (initial hits
1000 REM RUN THE COURSE
1010 FOR X=5 TO LEN C$-1 STEP 2           (scan course string
1020 SCROLL
1030 PRINT AT 21,P;"*";AT 21,P+G;"*";AT 21,S;"V"
1040 GOSUB 20                            (check for movement
1050 LET P=P+VAL C$(X TO X+1)            (update post position
1060 GOSUB 20                            (another quick check..
1070 IF S<=P OR S>=P+G THEN LET H=H+1 (has skier hit posts?
1080 NEXT X
2000 REM FINISHING LINE
2010 SCROLL
2020 FOR X=P TO P+G                      (print finishing line
2030 PRINT AT 21,X;"-";
2040 NEXT X
2100 SCROLL
2110 SCROLL
2120 PRINT "YOU HIT ";H;" POSTS"
2130 GOTO 9999
8000 REM SELECT COURSE
8010 REM SEE TEXT IN BOOK
8020 LET C$="141700000000-1-1-1-
     1-100000001010200-1-1-100010200-
     200-1-1-1000000000101010101-2-
     2-2000000000101010101010102020
     2000000000000-1-1-1010101-1-1-1-
     1-1-2-2-2-2-2-1-10000000101010
     101-2-2-2-200000001010202020201O
     101000000000000000000"
8030 IF RND<0.5 THEN RETURN
8040 LET C$="000200000000000000000
     0000000000101010101010101010100O
     000000202020200000-1-1-1-1-1-1-1-
     1010101010000000000000000-1-1-1-
     102020202010101-2-2-2-2-2-2-2-1-
     1-1-1-10101-1-1000000000000"
8050 RETURN
9000 REM RULES
9010 CLS
9020 SLOW
9030 PRINT TAB 8;"*** SKI RUN ***"
9040 PRINT
```

```
9050 PRINT " YOU MUST STAY IN BETWEEN THE",
           "COURSE POSTS ALL THE WAY DOWN.",
           " IF YOU HIT THE POSTS OR MOVE",
           "OUTSIDE THE COURSE, YOU WILL",
           "LOSE POINTS."
9060 PRINT " YOU MAY INITIALLY SELECT THE",
           "DIFFICULTY OF COURSE YOU WISH",
           "TO ATTEMPT."
9070 PRINT " USE THE ""5"" AND ""8"" KEYS TO",
           "DIRECT YOURSELF LEFT AND RIGHT."
9080 PRINT
9100 PRINT "WHICH LEVEL ARE YOU?",
           " (1) RANK AMATEUR",
           " (2) AVERAGE SKIER",
           " (3) DASHING EXPERT?"
9110 INPUT SKILL
9120 IF SKILL>0 AND SKILL<4 THEN GOTO 9200
9130 CLS
9140 PRINT "SORRY? - ";
9150 GOTO 9100
9200 LET G=3+(4-SKILL)*2                    (post width
9210 PRINT "ARE YOU READY? - ";
9220 FOR X=3 TO 1 STEP -1                   (give 3 sec. delay
9230 PRINT X;"..";
9240 PAUSE 50
9250 NEXT X
9270 RETURN
```

Course selection is performed in the subroutine starting at line 8000. You may prefer to ask for a course number to be entered as the program is started, or just replace the two courses given here. The rules for how to create your own courses follow:-

After subroutine 8000 has finished, string variable C$ must contain a course description in the following format:-

C$(1 TO 2)	starting position on screen of left-hand post (the right-hand post position is calculated from the degree of skill entered)

C$(3 TO 4)	starting position of skier - this would normally be the post starting position plus two. If you want to be annoying, you can start the player off <u>outside</u> the posts.
C$(5 TO)	the rest of the string is a series of two-digit numeric values which represent the value to add into the current post position to give the next position. Do not use values greater than "02" or less than "-2", as the player cannot move the skier more than two positions in any one "turn" (i.e. as each new set of posts is printed).

A small but valid course description would be:-

```
15 17 00 00 01 01 -1 -1 -2 -2 00 00
```

This says:-

The post starting position is column 15, and the player starts at column 17. On the first two "turns" the posts remain in the same position. The next two turns move the posts over to the right (positive value) by one position each time (01). The next two turns move the posts to the left by one position each turn (-1). The following two turns move the posts to the left by two (-2) positions, while the final two turns leave the posts in this position.

The program will automatically show a finishing line and the number of "hits" made.

Since the speed of the game is approximately 3 turns per second, an ideal course should take between 30 seconds and one minute to run fully. Of course, (no pun) you may prefer to enter a five minute course - there's no reason why you should not.

Beware that you keep your course within the boundaries of the screen, otherwise you will be given error B when the program runs. Your course should not go beyond column 22 or become negative. The reason for not using columns beyond 22 is that the width of the posts is 9 when an amateur is running the course.

EFFICIENT PROGRAMMING

In this section, we deal with efficient programming. This can be handled in two ways:-

1) optimising memory
2) optimising run-times.

Depending on the particular program, a decision may be needed to trade off memory usage over run-times. This type of trade-off is all too commom on small computer systems and even with the 16K RAM pack fitted, you may find that certain programs (e.g. a full version of Star Trek) still require attention to detail to fit in comfortably.

The Sinclair handbook gives quite explicit details of the way in which programs and variables are stored inside the ZX81. It may not be instantly apparent, however, that certain features can add quite an overhead into your programs.

Let's look at the way each statement is stored.

3.1 Replacing literals

Assuming you have a 1K RAM ZX81 (if you have more, then this section will not be quite so important to you), when you switch the ZX81 on, you can have up to 899 bytes (maximum) available since the system variables occupy 125 bytes - more than one tenth of the total memory!

The screen buffer requires at least 25 bytes, since space is taken as items are entered into the display buffer using PRINT or PLOT commands. This does not apply when the 16K RAM pack is fitted.

So you have roughly 850 bytes to share between program statements, variables, display (when used), and some assorted items used by the BASIC interpreter during the course of program execution.

Each program statement requires:-

- 2 bytes to hold the line number (regardless of the number of digits in the line number)
- 2 bytes to indicate the length of the statement
- the number of bytes in the statement itself (this can be easily calculated by the number of keystrokes made in typing the line - see below)
- an extra six bytes for every numeric literal entered
- one extra for a newline

Let's see a few examples of this:-

Statement	Size in memory
10 LET X=Y	9 bytes in total - 2 for line number, 2 for line length, 4 keystrokes (LET,X,=,Y), plus one for newline.
300 LET X=1	15 bytes in total - 2 for line number, 2 for length, 4 keystrokes (LET,X,=,1), 6 for the numeric literal "1", and one for newline.
9000 DIM X(3,2)	24 bytes - 2 for line no., 2 for length, 7 keystrokes (count them), 12 for the numeric literals "2" and "3", one for newline.

It should be rather obvious by now that the use of numeric literals in a program is extremely wasteful of memory.

Why was the BASIC interpreter designed in this way? One very good reason is that the conversion of literals into 5-byte floating point values (and vice versa) is a slow process - you may have noticed this when you PRINT some numeric expressions. This comes back to our trade-off that was mentioned above. All numeric literals are held in converted form in the program statements so that the conversion time is not required when the program is run. It was incurred as you entered the statement (this is part of the syntax checking routines).

You can make good use of this knowledge - try to keep the number of literals to a minimum. Your programs will become smaller, thus leaving you room for a slightly larger program to fit in.

There are several ways that you can use in order to avoid using literals - some cause the program to run more slowly, some are just more efficient ways of writing the program that may seem odd to someone else.

One method is to put commonly used values into a variable that is used instead of the literal value, for example:-

```
10 IF X<>1 THEN LET A=1        (26 bytes
20 IF Y<>1 THEN LET B=1        (also 26 bytes
```

...could be written as:-

```
 5 LET U=1                     (15 bytes
10 IF X<>U THEN LET A=U        (14 bytes
20 IF Y<>U THEN LET B=U        (also 14 bytes
```

The first method occupies 52 bytes of memory, while the second only occupies 43 bytes - a saving of 9 bytes even though there is an extra statement in the second program!

Unfortunately, this is not strictly accurate, as the second program requires a variable U which is not used in the first program, and a simple numeric variable like U will occupy 6 bytes. The total saving is now 3 bytes - not a fortune, but every time that the literal "1" is used in the same program, it can be altered to use variable U and save five bytes each time.

Another advantage of this method is that it does not slow the program down in any noticeable way - try this benchmark to see how much effect it has:-

```
BM1A        10 LET Y=0
            20 FOR T=1 TO 1000
            30 LET X=27
            40 NEXT T
          9999 STOP

BM1B        replace line 30 with 30 LET X=Y
```

If you prefer not to try these benchmarks (and those that follow), I have included a summary of the run times at the end of the book.

Run the first program (BM1A) and time it - don't worry that it takes a minute or two. When the message 9/9999 appears, stop timing. Now run the second program - BM1B - alter line 30 as instructed. Time it in exactly the same way. The difference in the times is the time it takes to search for variable Y and assign it to X as opposed to using the literal value held inside line 30. Since the program loops 1000 times, the error is multiplied by 1000.

This will give you an idea of how little difference there is between the two methods.

There is no need to assign a special variable for either 0 or 1, as these values can be obtained by using another variable that already exists in the program.

Suppose your program has a variable D in use. The expression:-

D=D	will give the value 1 (a number <u>must</u> be equal to itself, and a "true" expression has the value 1).
NOT D=D	will give the value zero, since this is only the reverse of the logic above.

Both of these occupy less memory than just writing the value 0 or 1 in a program statement, although they will be slightly slower. If your program requires speed, then use literals. Try these benchmarks to see what effect the expressions have:-

```
BM2A        10 LET Y=20
            20 FOR T=1 TO 1000
            30 LET X=1
            40 NEXT T
          9999 STOP

BM2B        replace line 30 with 30 LET X=Y=Y


BM3A        replace line 30 with 30 LET X=0

BM3B        replace line 30 with 30 LET X=NOT Y=Y
```

Time the programs as you did earlier and check the results for parts A and B of each program benchmark. The results should speak for themselves - the "B" version of each program occupies less memory: what difference does it make to the run time?

Now you are in a position to choose the method that a particular program needs - speed or memory.

The statement LET U=1 requires 15 bytes and variable U also uses six bytes. Each time that you now use variable U instead of the literal "1", you are saving 6 bytes, so there is no point in altering a program unless you can replace four literals, since 4*6 = 24 bytes saved, while you have added 21 bytes by writing LET U=1.

You can see that each program needs to be carefully considered - is it actually saving memory, or is it going to increase?

You may like to study the "Dice Rolling" program - this has been written with memory conservation in mind.

3.2 Avoiding numeric values

On many occasions, a program requires the use of numeric values, but does not really need the glory of 5-byte floating point! In the 1K RAM ZX81, these 5-byte variables can occupy a large amount of space - especially when literals are held within the program statements (see above).

A typical program which requires some preset values would look like :-

```
 10 DIM V(10)
 20 LET V(1)=22
 30 LET V(2)=9
 40 LET V(3)=11
 50 LET V(4)=17
 ....
 ....
110 LET V(10)=3
```

From the previous section, you know that each of those statements costs at least 24 bytes, a luxury that is hard to afford. You could always enter these as direct commands, then save the program with array V containing preset values, but sometimes even this method is unattractive. A preferable solution is to set these values into a string, then use the **VAL** function to convert them. The numbers can be much shorter, as you only need to hold (say) a two digit string for each value.

Look at this:-

```
10 DIM V(10)
20 LET V$="22091117036651042003"       (31 bytes
30 FOR V=0 TO 9                        (23 bytes
40 LET V(V+1)=VAL V$(V*2+1 TO V*2+2)   (57 bytes
50 NEXT V                              (7 bytes
```

The overall saving here is approximately 100 bytes - enough to hold quite a few more program statements.

You can see this type of routine used in the program "Ski Run".

If the string is particularly long (in the example above, the string in V$), you can release the space from variables by declaring the variable as the empty string after the routine has completed. Using the example above, this would mean adding one extra program statement:-

```
60 LET V$=""
```

Again, each program has different needs and it is almost impossible to make a general purpose subroutine that is also extremely efficient. I have attempted to highlight the various areas where you can avoid wasting memory and to show you a possible way of resolving the problem. The actual program statements that you enter in your programs may look very different from those given above. If you wish to use a fairly generalised routine, I have included one below.

You may be interested to see if using a string to hold a numeric value is any faster or slower than using a numeric literal. Try this comparison benchmark:-

```
BM4A        10 LET X$="99"
            20 FOR T=1 TO 1000
            30 LET X=1
            40 NEXT T
          9999 STOP

BM4B        replace line 30 with 30 LET X=VAL X$
```

Notice that the time taken for program BM4A should correspond (almost) to the time for programs BM1A, BM2A and BM3A.

3.3 String Data Storage

Some versions of Basic use the commands DATA and READ to allow numbers and strings to be conveniently stored within a program. Although it is not a problem to do without these commands, there are times when it would be helpful to just store a string of words or numbers in a program without going to the expense of a string array.

If a set of words of varying lengths is to be stored, a string array must always be set up to the length of the longest item - this can waste a large amount of memory even though it is quick to process, as only a subscript number is needed to give the complete string.

Where speed is not particularly important, or space is important, you may like to use this routine (or one similar) that will pick out a single word or phrase from an ordinary string variable.

The routine is used in the program "Tunnels and Trolls Character Generator" which follows later on in the book.

If you wish to store numeric values, then you only need to apply a **VAL** function to the string that is returned from this subroutine.

The following variables are needed to work the routine:-

D$	which contains the string of words. Each word should be separated by "/" characters. You can change this by altering lines 9030 and 9050 below.
N	which contains the word number which you want to be extracted from D$.
W$	is set to the word requested, or the empty string (i.e. length zero) if word number N cannot be found in the string D$. Apply a **VAL** function to this variable if you wish for numeric values to be held.

Tape name: "READ$"

```
9000 REM PUT NTH WORD FROM D$ INTO W$
9005 LET XW=0
9010 LET W$=""
9015 FOR X=1 TO N-1
9020 LET XW=XW+1
9025 IF XW>LEN D$ THEN RETURN
9030 IF D$(XW)<>"/" THEN GOTO 9020
9035 NEXT X
9040 LET XW=XW+1
9045 IF XW>LEN D$ THEN RETURN
9050 IF D$(XW)="/" THEN RETURN
9055 LET W$=W$+D$(XW)
9060 GOTO 9040
```

You may well be thinking that this routine is not going to save much memory space. Obviously, such a general routine would be used where large quantities of items are to be stored - you would probably omit the REM statements in any such routine as well.

The total length of the routine above (including all the special variables it uses, but not including the REM) is 222 bytes. This implies that a 16K RAM pack is really necessary to gain full benefits of such a method - although there are still 500-odd left in a 1K ZX81, so don't rule it out completely!

Remember that the string variable used can be assigned by a direct command and then stored on cassette tape with the program. This avoids holding a lengthy LET statement in the program.

3.4 Program Structure

The ZX81 may seem quite fast to people who do not have any real involvement with computers but, as good (and cheap) as it is, it is also quite slow. When compared to not only larger commercial business computers but also other personal computers, the ZX81 appears slow. This can make interactive programs slow to respond to any key being pressed.

In the discussions above, you have seen how a program can be made more efficient in terms of memory usage; now we turn to speed.

First of all, consider frequently-used subroutines. In any program, whenever a GOTO or GOSUB statement is obeyed, the ZX81 searches for the relevant line number by scanning the program from the lowest line number.

If your program requires speed (and any small improvement helps), then ensure that all the commonly-used subroutines are kept at the beginning of the program (see the "Ski Run" program). Your program layout would look something like:-

```
   1 REM program name etc
  10 GOTO 1000
  20 REM frequent subroutines here
  ....
  ....
1000 REM main program
  ....
  ....
8999 STOP
9000 REM infrequent subroutines
         (initialisation etc)
```

The following benchmark programs will give you some insight into the way that various time savings can be made by writing certain statements in different ways:-

```
BM5A      10 LET X=1
          20 FOR T=1 TO 1000
          30 IF X=1 THEN NEXT T
          40 NEXT T
        9999 STOP

BM5B      replace line 30 with  30 IF X=0 THEN NEXT T

BM6A      replace line 30 with  30 IF X THEN NEXT T

BM6B      replace line 30 with  30 IF NOT X THEN NEXT T
```

The result of these tests shows that it is (slightly) more efficient to ensure that a condition will normally fail (i.e. be "false"). Obviously, if you need to introduce extra program statements in order to make sure that a condition fails, then it becomes slightly self-defeating. However, if you are in a position to choose, then try to make sure that any conditions in your programs are set up such that in the majority of cases, the condition will result in a "false" evaluation.

The other conclusion that can be drawn from these last two benchmarks is that it is more efficient, both in speed and memory usage, to test a true or false value directly (as in BM6A and BM6B) without using an expression such as X=1 or X=0. This can only be used when the variable value is zero and non-zero (since "true" is taken to be a non-zero value).

Study the results carefully - your programs can benefit considerably from this type of information.

One other comparison for you - many programs use a variable which increments by one each time around the main loop. This can often be replaced by a FOR/NEXT loop, but is it worth it? Try this:-

```
BM7A        10 LET T=1
            20 IF T=1000 THEN STOP
            30 LET T=T+1
            40 GOTO 20

BM7B        10 FOR T=1 TO 1000
            20 NEXT T
          9999 STOP
```

I'll let you see for yourself which method is better.

3.5 Expression Values

Expression values can be extremely useful in making a program more compact. They may or may not speed it up, and they can certainly make it hard to understand a program, but used with care, complex expressions can reduce memory overheads quite significantly.

You are probably aware by now that the ZX81 reduces all conditional expressions to the values 0 (if the expression is false) or 1 (if true).

A common use of this feature is when one variable is to be assigned with a different value depending on the value of one or more different variables. An example of this can be found in the program "Etch-a-Sketch". The lines are reproduced below:-

```
110 LET DX=(D=5)*-1+(D=8)
120 LET DY=(D=6)*-1+(D=7)
```

Depending on the value of variable D, variables DX and DY are set to one of four possible combinations:-

Value of D	Result in DX	Result in DY
5	-1	0
6	0	-1
7	0	+1
8	+1	0

This could have been written:-

```
110  LET DX=0
114  LET DY=0
118  IF D=5 THEN LET DX=-1
122  IF D=6 THEN LET DY=-1
126  IF D=7 THEN LET DY=1
128  IF D=8 THEN LET DX=1
```

...but it is quite easy to see that this solution occupies considerably more memory (142 bytes, while the original lines occupy 82 bytes!).

Since variable D can only have one value at any one time, stating "(D=5)" can either be true (value 1) or false (value 0). Multiplying this result by any other number, n, gives either n if the expression was true, or zero if the expression was false.

This can be put to another memory-saving use. Suppose a program wishes to test the variable:-

```
        IF A$(1)="Y" THEN ....
```

If the program needs this expression more than once, it will be less costly to write:-

```
        LET A=(A$(1)="Y")
```

...and then test variable A for being true or false:-

```
        IF A THEN ...
```

It is hopefully obvious that variable A reflects the result of the test at the time of assigning the variable. If A$ should change, variable A will not necessarily give the correct result unless the assignment is made again.

This can therefore be put into a complete subroutine. A typical use of this type of routine can be seen when a "yes/no" reply is entered into a program:-

```
8000 REM GET YES/NO INPUT
8010 PRINT "ANSWER YES OR NO"
8020 INPUT Y$
8030 LET Y=CHR$ CODE A$="Y"
8040 RETURN
```

This particular routine is not very rigid in the checks made on the information entered in line 8020, although you could alter this as you wish. The point is that variable Y will now reflect the answer YES or NO by simply stating:-

```
    IF Y THEN PRINT "YOU ANSWERED YES"        (or whatever)
  or
    IF NOT Y THEN PRINT "YOU TYPED NO"
```

Although this is quite an expensive subroutine to use if the variable Y is only used once or twice in a program, it can quickly save memory (and program speed - compare the results of programs BM5 and BM6) if used several times, since the full expression has already been evaluated.

Note for ZX80 users

With the original 4K ROM, the AND and OR operators used boolean logic since all arithmetic was integer (2-byte). The same "tricks" cannot, however, be used on the 8K ROM, as the AND and OR operators now handle 5-byte arithmetic and give a value of 1 for "true" instead of -1. Notice also that -1 does not mean "all bits set" any longer.

*** Stocking Filler ***

Suitable for : 1K RAM

Try to stop the moving star by pressing one of the "number" keys (1-9). The lower the score, the better. It is easy to stop it by pressing "9", but you should try to keep the score low by pressing "3" or "2". If you miss with one key, try another higher valued key. This is not really suitable for the ZX80.

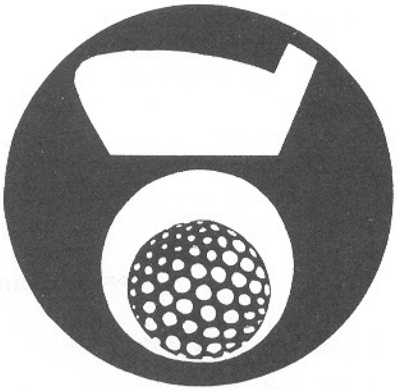
```
 10 PRINT AT 10,0;"*"
 20 FOR X=1 TO RND*50+25
 30 NEXT X
 40 FOR X=1 TO 10
 50 PRINT AT 10,(X-1)*3;" ";AT 10,X*3;"*"
 60 LET Z=CODE INKEY$
 70 IF Z AND Z-28>=X AND Z-28<10 THEN GOTO 110
 80 NEXT X
 90 PRINT AT 14,10;"FAILED"
100 GOTO 120
110 PRINT AT 14,10;"SCORE:";Z-28-X
120 PAUSE 4E4
130 IF INKEY$<>"" THEN GOTO 130
140 CLS
150 RUN
```

Pro-Am Golf Putting

Suitable for : 16K RAM

You've managed to reach the green, now it's time to putt the ball. The hole is represented by an inverse "O", while the ball is shown as an inverse "X". You need to enter the direction and strength required to sink the ball.

This program uses the "line plotting" routine given in the ZX81 handbook, although a couple of minor changes have been made to allow the routine to blank out a line as well as draw one.



Tape name: "PUTTING"

```
   1 REM *** GOLF PUTTING ***
  10 GOSUB 4000                           (print instructions
  20 CLS
  30 REM DRAW BORDERS
  40 FAST
  50 GOSUB 2000
 100 LET HOLEX=INT (RND*30)+1             (place pin
 110 LET HOLEY=INT (RND*18)+1
 120 LET TEEX=INT (RND*30)+1              (place tee
 130 LET TEEY=INT (RND*18)+1
 140 IF TEEX=HOLEX AND TEEY=HOLEY THEN GOTO 120
 150 LET HITS=0
 190 SLOW
 200 PRINT AT HOLEY,HOLEX;"O"             (use inverse "O"
 210 PRINT AT TEEY,TEEX;"X"               (use inverse "X"
 300 REM MAIN LOOP
 310 GOSUB 3100                           (clear line 21
 320 PRINT "DIRECTION?"
```

```
330 INPUT XD
340 GOSUB 3000                          (remove prompt
350 IF XD<0 OR XD>12 THEN GOTO 300      (check direction
360 PRINT TAB 22;XD
370 GOSUB 3000
380 PRINT "STRENGTH?"
390 INPUT XS
400 LET XS=INT XS                       (strength is integer
410 GOSUB 3000                          (clear prompt
420 PRINT TAB 28;XS
500 LET A=TEEX*2                        (prepare to draw line
510 LET B=INT ((TEEY*(-2))+42)
520 LET XD=(XD*(-1))+15                 (calculate true direction
530 IF XD=12 THEN LET XD=0
540 LET C=A+INT (COS ((PI/6)*XD)*XS)    (destination
550 LET D=B+INT (SIN ((PI/6)*XD)*XS)    (     "
560 LET XA=A
570 LET XB=B
580 GOSUB 1500                          (check new tee location
590 LET P=1                             (set "plot" marker
600 GOSUB 1000                          (plot the line
610 PRINT AT TEEY,TEEX;" ";             (blank out old tee
620 LET TEEY=INT ((D-42)/(-2)+0.5)      (calculate new tee-off
630 LET TEEX=INT ((C+0.5)/2)
640 LET HITS=HITS+1
700 IF TEEX=HOLEX AND TEEY=HOLEY THEN GOTO 800
                                        (has ball reached hole?
710 LET A=XA                            (now "unplot" line
720 LET B=XB
730 LET P=0                             (set "unplot" marker
740 GOSUB 1000                          (undraw the line
750 GOTO 200
800 GOSUB 3400                          (print "plop"
810 PRINT AT 20,0;"YOU DID IT"          (winning messages
820 PRINT "IT TOOK ";HITS;" PUTTS";
830 GOSUB 3500                          (print rating
840 IF INKEY$ ="" THEN GOTO 840         (wait for another go
850 CLEAR
860 GOTO 20
```

```
1000 LET U=C-A                              (see Sinclair book
1010 LET V=D-B
1020 LET D1X=SGN U
1030 LET D1Y=SGN V
1040 LET D2X=SGN U
1050 LET D2Y=0
1060 LET M=ABS U
1070 LET N=ABS V
1080 IF M>N THEN GOTO 1130
1090 LET D2X=0
1100 LET D2Y=SGN V
1110 LET M=ABS V
1120 LET N=ABS U
1140 LET S=INT (M/2)
1150 FOR I=0 TO M
1160 IF P THEN PLOT A,B
1165 IF NOT P THEN UNPLOT A,B
1170 LET S=S+N
1180 IF S<M THEN GOTO 1230
1190 LET S=S-M
1200 LET A=A+D1X
1210 LET B=B+D1Y
1220 NEXT I
1230 LET A=A+D2X
1240 LET B=B+D2Y
1250 NEXT I
1260 RETURN
1500 REM CHECK NEW TEE LOCATION
1510 LET ERROR=(C<2 OR C>62 OR D<6 OR D>42)
1520 IF NOT ERROR THEN RETURN
1530 PRINT AT 21,0;"INTO THE ROUGH - PENALTY HIT"
1540 LET HITS=HITS+1
1550 IF C<2 THEN LET C=2
1560 IF C>62 THEN LET C=62
1570 IF D<6 THEN LET D=6
1580 IF D>42 THEN LET D=42
1590 RETURN
2000 REM DRAW GREEN LIMITS
2010 LET Y=43
```

```
2020 GOSUB 2100
2030 LET Y=4
2040 GOSUB 2100
2050 LET X=0
2060 GOSUB 2200
2070 LET X=63
2080 GOSUB 2200
2090 RETURN
2100 REM DRAW "X" LINE
2110 FOR X=0 TO 63
2120 PLOT X,Y
2130 NEXT X
2140 RETURN
2200 REM DRAW "Y" LINE
2210 FOR Y=4 TO 43
2220 PLOT X,Y
2230 NEXT Y
2240 RETURN
3000 REM CLEAR PROMPTS
3010 PRINT AT 20,0;"   <-20 spaces->     "
3020 PRINT AT 20,0;
3030 RETURN
3100 REM CLEAR LINE 20
3110 PRINT AT 20,0;"   <-63 spaces->     "
3120 PRINT AT 20,0;
3130 RETURN
3400 REM PLOP
3410 IF HOLEX<2 THEN LET HOLEX=2
3420 IF HOLEX>29 THEN LET HOLEX=29
3430 PRINT AT HOLEY,HOLEX-2;"PLOP"
3440 RETURN
3500 REM SILLY MESSAGES
3510 IF HITS=1 THEN PRINT " - HOLE IN ONE";
3520 IF HITS=2 THEN PRINT " - NOT BAD...";
3530 IF HITS<6 THEN GOTO 3590
3540 IF HITS<8 THEN PRINT " - ONLY FAIR";
3550 IF HITS>7 THEN PRINT " - DISGUSTING.";
3590 PRINT
3595 RETURN
```

```
4000 REM INSTRUCTIONS
4010 SLOW
4020 PRINT TAB 8;"GOLF PUTTING"
4030 PRINT ,,"DO YOU WANT INSTRUCTIONS?"
4040 RAND
4050 INPUT Y$
4060 IF CHR$ CODE Y$<>"Y" THEN RETURN
4100 PRINT ,," THE TEE IS SHOWN AS X AND THE",
          "PIN AS O. YOU MUST JUDGE THE",
          "DIRECTION AND STRENGTH NEEDED",
          "TO SINK THE BALL."
4110 PRINT " THE DIRECTIONS ARE 1 TO 12 (AS",
          "ON THE HANDS OF A CLOCK) BUT",
          "YOU MAY USE FRACTIONS, LIKE 2.6",
          "OR 10.1."
4120 PRINT " IT TAKES A STRENGTH OF 60 TO",
          "HIT THE BALL FROM ONE SIDE TO",
          "THE OTHER, AND 36 TO HIT FROM",
          "TOP TO BOTTOM."
4130 PRINT " WHEN A GAME IS OVER, PRESS ANY",
          "KEY TO START ANOTHER."
4140 PRINT ,,"HAVE FUN."
4150 PRINT ,,"(PRESS A KEY)"
4160 PAUSE 4E4
4170 POKE 16437,255
4180 RETURN
9900 SAVE "PUTTING"
9910 RUN
```

Here are the changes you'll need to run this program on a ZX80:-

```
605 PAUSE 150

840 PAUSE 4E4
841 POKE 16437,255
```

EYEBALL CHARACTERS

4.1 What it does

This routine allows you to print a string in large 8-by-8 format across the screen. It uses the PLOT command to draw the letters, so you can fit eight characters across the screen. Since the depth of the screen is only 44 pixels, you can have up to 5 lines (allowing you one blank "pixel line" in between).

It's an impressive way of introducing a program when it's loaded - the name of the program can be made to appear in huge characters on the screen!

4.2 How it works

The subroutine makes use of the 8K ROM contained in the ZX81 - a character generator is held at address 7680 onwards. If you're not familiar with more detailed workings of a Z80 microcomputer, then you may prefer to skip the next few paragraphs since it's not really necessary to understand how the routine works, although you may be interested.

The character generator is a section of memory that contains the pattern of each character that is output to the display. Each displayed character occupies 8 bytes of memory, where each byte corresponds to the pattern of one row of the character on the screen. As an example, take the character "A". The address of the generator pattern for "A" is 7984 (more on this below), and the eight bytes starting at this address contain:-

Eyeball Characters

address	contains		gives on screen
	decimal	binary	
7984	0	00000000	....
7985	60	00111100	..****..
7986	66	01000010	.*....*.
7987	66	01000010	.*....*.
7988	126	01111110	.******.
7989	66	01000010	.*....*.
7990	66	01000010	.*....*.
7991	0	00000000	....

You can see the letter "A" taking shape in the last column. Each character's pattern is found at the address:-

7680 + (character code * 8)

...but be warned that only characters with a code in the range 0 to 63 are valid. All the other codes (greater than 63) are either non-existent, or use multiples of these characters: for example the keyword RND has a code 64, yet it is displayed by the combination of the letters R, N and D.

4.3 The subroutine

So what about the subroutine? It uses the character generator above to plot pixels on the screen. Since the generator occupies an 8X8 grid, it's fairly easy to see how this can be enlarged.

One problem to be overcome, though, is how to look at each bit in the contents of the generator, since the routine needs to plot a pixel when a bit is set, and skip over a pixel if a bit is unset (as shown above by using * to represent a bit set, and full-stop to indicate a bit unset).

I have used two versions of this routine - neither is particularly fast, but the second is at least noticeably faster than the first. The first method is slightly easier to follow, though, as it uses the more conventional approach to isolating a single bit from a byte.

The routine requires three variables to be set up before it is called (further on you will see a full example of the routine in use within a program) -

A$	must contain a string of characters to be displayed in large format. Maximum of eight characters (else the ZX81 will stop with error report B). The characters can only have codes in the range 0 to 63.
XX	the pixel position (across the screen) of the first character in the string.
YY	the pixel position (down the screen) of the first character in the string. Note that 43 is the <u>top</u> of the screen, while 0 is the bottom.

Version 1 - easy(?) to understand, but slower

```
9300 REM PLOT A$ INTO BIG CHARACTERS
9305 FOR A=1 TO LEN A$                  (for each character
9310 LET XC=CODE A$(A)                  (get character code
9315 GOSUB 9340                         (print in 8X8 grid
9320 LET XX=XX+8                        (next position on screen
9325 NEXT A                             (next letter
9330 RETURN                             (exit - all done
9340 FOR Y=YY TO YY-7 STEP -1           (eight rows down screen
9345 LET ROW=PEEK (7680+8*XC+YY-Y)      (get row pattern from ROM
9350 LET SHR=128                        (bit mask initial value
9355 FOR X=XX TO XX+7                   (for each bit in row
9360 LET Z=INT (ROW/SHR)                (put bit setting in Z
9365 LET ROW=ROW-Z*SHR                  (take this bit out
9370 LET SHR=SHR/2                      (update mask for next bit
9375 IF Z THEN PLOT X,Y                 (plot if bit was set
9380 NEXT X                             (next bit pattern
9385 NEXT Y                             (next row
9390 RETURN                             (character complete
```

Version 2 follows exactly the same principle, but since calculations on the ZX81 are slow (5-byte floating point is a bit like overkill on occasions when most programs can get by with integers!), this method uses strings to hold as much as possible. The bit mask is no longer held as a numeric variable, but by a string whose codes represent the bit masks 128,64,32,16,8,4,2,1.

This also means that most of the tests can be made on strings rather than numbers, although this doesn't actually speed things up.

Version 2 - faster but trickier

```
9300 REM PLOT A$ INTO BIG CHARACTERS
9305 LET Z$="        "          (see below
9310 FOR A=1 TO LEN A$
9315 LET XC=CODE A$(A)
9320 GOSUB 9340
9325 LET XX=XX+8
9330 NEXT A
9335 RETURN
9340 FOR Y=YY TO YY-7 STEP -1
9345 LET X$=CHR$ (PEEK (7680+8*XC+YY-Y))
9350 FOR X=1 TO 8
9355 IF X$<Z$(X) THEN GOTO 9370
9360 PLOT XX+X-1,Y
9365 LET X$=CHR$ (CODE X$-CODE Z$(X))
9370 NEXT X
9375 NEXT Y
9380 RETURN
```

Line 9305 above sets up string Z$ to the bit mask string. The characters needed are:-

128	inverse space	GRAPHICS/SPACE
64	**RND**	FUNCTION/T
32	4	4
16	(	(
8	grey square	GRAPHICS/SHIFT/A
4	quarter square	GRAPHICS/SHIFT/4
2	" "	GRAPHICS/SHIFT/2
1	" "	GRAPHICS/SHIFT/1

Although the string may look a bit odd, it is not used in a conventional way, but rather lets us get over the problem of not having any boolean operators or integer variables.

4.4 An example of use

So how do you use this? Here's an example of the routine in use. It prints the message HELLO in jumbo letters in the centre of the screen:

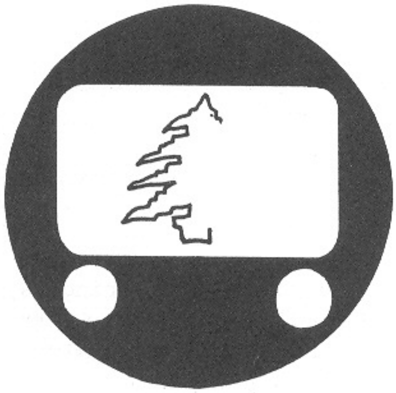
Tape name: "EYEBALL"

```
100 REM PRINT BIG HELLO
110 LET A$="HELLO"                (message to print
120 LET XX=12                     ("across" position
130 LET YY=26                     ("down" position
140 GOSUB 9300                    (invoke subroutine
150 STOP                          (see what you've done
```

Obviously, your program must incorporate one of the subroutines above (preferably the second) in order to work.

Paint-a-Pic

Suitable for : 1K RAM



Perhaps you had one of these types of toys when you were younger - I certainly did. This version does not easily draw "bends" (neither did the toy), but gives you some interesting patterns.

Use the cursor keys (without SHIFT) to draw a line in any direction. While you press the key, the picture is drawn. At any time, you may press the "0" key (labelled RUBOUT) to clear the screen. This leaves the marker at the same position, so if you want to draw something starting from the bottom right-hand corner, first draw a line to the starting position, then clear the screen by pressing "0".

Line 30 is quite interesting. It looks at the system variable RAMTOP to see if the 16K RAM pack is fitted. Without the RAM pack, this is set to 68 (68*256=17408, which is the address of the top of memory - 16384+1024). If any value greater than 68 is found, then extra memory must be fitted (either a 16K or 4K RAM pack - the 4K RAM packs were only available in the early ZX80 days). If extra memory is available, then the whole screen can be used for the picture, otherwise a restricted boundary must be set up to avoid error 4 occurring.

Tape name: "ETCH"

```
 10 LET MX=64                          (set max screen size
 20 LET MY=44                          (in X and Y directn
 30 IF PEEK 16389<69 THEN LET MY=23    (see if 16K fitted
 40 LET X=NOT MX                       (starting point = 0.
 50 LET Y=MY-1                         (  "          "  = MY
100 LET D=CODE INKEY$-28               (get keycode
110 LET DX=(D=5)*-1+(D=8)                          (calculate X directn
120 LET DY=(D=6)*-1+(D=7)                          (calculate Y directn
130 IF NOT D THEN CLS                              (check for clear
140 IF INT ((X+DX)/MX) OR INT ((Y+DY)/MY) THEN GOTO 200
                                                   (check if on screen
150 LET X=X+DX                                     (update X position
160 LET Y=Y+DY                                     (update Y position
170 PLOT X,Y                                       (draw new character
200 GOTO 100                                       (keep going...
```

If you have a ZX80 with the 8K ROM, you will need to add three lines, since it is not possible to watch the screen while a program is running.

Add :-

```
135 IF NOT DX AND NOT DY THEN GOTO 210    (no key pressed

210 PAUSE 4E4                             (display screen
220 GOTO 100
```

Alter line 30 to read:-

```
 30 IF PEEK 16389<69 THEN LET MY=22
```

This means that while you press a key, the screen will go blank, but once you lift your finger, the screen will appear, showing you your beautiful handiwork!

HINTS 'N' TIPS

As with most other people, I have gradually built up a series of small useful (?) items that have nothing other than a sense of beauty and simplicity about them. They are not really useful programming aids, nor are they earth-shattering memory savers - just simple ideas that can make programming more pleasurable, especially on the ZX81.

5.1 Pause Forever

The first item is one I find particularly neat, and you will probably notice it used throughout the programs in the book.

According to the Sinclair book, when the PAUSE command is used with a value greater than 32767, it is treated as "pause forever". The only way to restart a program in such a state is by pressing a key.

This is extremely useful, particularly for the ZX80 users who have bought the 8K ROM expansion, since the ZX80 cannot display the screen while a program is running, so the INKEY$ function cannot be used to its full potential. This is not relevant to the ZX81 unless running in fast mode.

A program would therefore contain a statement such as :-

```
200 PAUSE 33000
```

As an aide-de-memoire (pardon my Greek), I use the expression:-

```
200 PAUSE 4E4
```

...for two reasons. Not only does it occupy slightly less memory, but it also sounds like "pause forever" if you pronounce it as "pause for-ee-for".

Remember that scientific notation is quite legitimate on the ZX81 and that 4E4 actually represents 40000, which is greater than 32767.

5.2 Ending a Program

This is another hobby-horse of mine - I like to see a program which finishes with a recognisable completion message, rather than something like 0/375 at the foot of the screen.

For this reason, I include the line:-

```
9999 STOP
```

...at the end of a program, and whenever the program terminates, all I need to write is...

```
... GOTO 9999
```

When the program reaches this line, it displays 9/9999 at the foot of the screen, which is always instantly recognisable as "completed O.K.". Any other error reports stand out quite easily when you are watching for 9/9999.

Obviously, many interactive programs are only stopped by pressing the BREAK key, but this tends to give an error D which you should be expecting.

5.3 REM statements

Having been involved in programming for more years than I care to put in print, I have met many different programs along the path.

One of the worst problems of taking on another person's programs is that of understanding the structure of them. This can make you waste large amounts of time trying to "unpick" a complex portion of programming.

I suspect that the vast majority of programs you write will be thrown away or replaced by newer and better versions. This type of program, which invariably serves to test an idea out, is not worth spending any amount of time on, but when you write a program that you intend to keep, sell or give to friends, it becomes (I feel) extremely important to spend some time considering the way you have made it easy for other people to follow.

The REM statement allows you to annotate various portions of a program so that it can be easily understood. You should try to use a REM statement:-

a) at the beginning of every subroutine, stating the purpose of the subroutine,

b) at each logical "section" of the program, such as the main loop, the initialisation section, the finishing section, etc.,

c) before any particularly complex passages of program, briefly outlining the intention of the routine.

Here comes the rub. On the 1K ZX81, there is not really enough memory to allow this luxury, and most of the programs need to use rather devious programming to fit in. I have tried to avoid this problem by putting comments alongside the program statements throughout the book, but if your program has room for a REM, try to put one in.

When the 16K RAM pack is fitted, space becomes much less important in programming considerations. You can afford to be rather extravagant with the REM command - after all, it only takes you a few seconds to type them and they have no effect on the way a program runs.

(Note that in most of the programs in this book, I have kept the REM statements to a minimum. This is because not only have I put comments alongside each program, but I also want to keep the amount of typing you have to do down to a reasonable minimum.)

5.4 Cassette Tapes

In case you are not aware, you can buy C12 tapes from most computer stores (also mail order). These tapes are much more convenient to use than the traditional C60 or C90 tapes.

It can become extremely irritating to sit and wait for the cassette to be wound from one end to the other, thus the shorter the tape, the less time you sit and wait.

You may be tempted to put all your programs onto one tape. Try to resist this temptation - three or four programs per side is really quite sufficient, otherwise you'll be waiting for a significant amount of time if the program you wish to load is the last one on the side.

It looks much more impressive to see a well-catalogued set of tapes than to watch someone fumbling around with one tape, making constant excuses such as "This won't take a minute" or "Nearly there!".

The original ZX80 did not allow a program to be saved or loaded by name, making it very awkward to combine several programs onto one tape. For this reason, ZX80 owners were advised to keep each program on a separate side of tape. Those who have bought the 8K ROM expansion also have the "named program" facility and will be able to "batch" their programs onto one tape with more confidence.

5.5 Saving Variables

If you decide to save a program with some preset variables, then include a "load-and-go" routine in the program so that it automatically runs when it is loaded from tape. This way, you will avoid typing RUN by mistake and clearing all the variables that you've so carefully preserved.

An example of a "load-and-go" routine can be found on page 110 of the Sinclair book. You will also find one in the "Adventure Loader" program in this book.

5.6 Loading Programs

In the early ZX80 days, a large amount of adverse comment was bandied about regarding the problems of loading programs. Some people found it extremely erratic, while others experienced no problems at all.

As always, provided you are careful to ensure that you abide by the rules (see Chapter 16 of Sinclair's book), you should find that once you have got it working, it will stay that way.

One item is missing from the list, and in producing cassette tapes of the programs contained in this book, we have found that using stereo recording equipment can (but not always) cause trouble.

If the heads are misaligned on a stereo recorder, then phase shift can occur between the two channels when played back on either a different recorder or even your own if the alignment is very bad. You will not hear anything wrong, as your ears will not detect a slight phase shift, but your ZX81 will certainly register a problem!

The solution is simple - if you are using stereo equipment, record in one channel only (preferably the left channel, since it is the outside channel and therefore reduces crosstalk). You may well find that it becomes easier to read your tapes on a cheap portable recorder.

5.7 Looking at strings

The ZX81 BASIC is fairly complete, but does suffer from one or two minor drawbacks. One of these is the irritating habit of error report 3 occurring when a program tries to look at a "slice" of a string, e.g.:

```
LET A$="ABCDE"
PRINT A$(6)
```

...will give error 3 since there are only five characters in A$. Personally, I would have preferred to see an empty string given, but I mustn't grumble too much!

If your program only needs to look at the first character of a string, then error 3 can be avoided by writing:-

```
CHR$ CODE A$
```

...since the **CODE** function supplies the code value of the first character, or zero if the string is empty. This way, the program does not need to include a line such as:-

```
IF LEN X$=0 THEN ....
```

So if you are looking at individual characters of a string, you can write:-

```
CHR$ CODE X$(n TO )
```

...which will never give error 3 while n>=1.

5.8 Saving memory

You may have caught onto this hint while reading the section "Efficient Programming", but it's still worth making it more explicit.

At any time, you can rewrite a statement such as:-

```
IF xxxx <> 0 THEN ....
```

...by stating instead...

```
IF xxxx THEN ....
```

...since a non-zero value is taken by the ZX81 to mean "true".

By the same logic, a statement such as:-

```
IF xxxx = 0 THEN ....
```

....can be written as...

```
IF NOT xxxx THEN ....
```

Both of these alternative methods are not only faster to run, but they occupy less memory.

It is important to be aware that you are not out to win a prize for writing the most devious piece of programming, but that you are attempting to make the most of the ZX81's features. A different computer system may not be so happy to understand a statement like those above, so you should be wary of assuming that your programs will now run on any computer system.

5.9 Inputting Expressions

It is very easy to overlook a quite powerful feature of the ZX81 - the fact that any valid expression (of the type requested) can be entered when an INPUT command is being executed.

This is put to good use in the program "Standard Deviation", where a list of data points is entered and terminated by typing END. Yet the program is expecting a numeric value?!

Since the variable END has already been assigned within the program (it is given the value 1E38, a value unlikely to be entered in the normal course of data entry), it is accepted by the INPUT statement as a legitimate numeric value.

You could also enter a value such as 25*3.555/6 - the ZX81 is just as happy.

The same applies to string input as well - any previously assigned string variables can legally be entered when an INPUT statement asks for some data. In this case, however, you need to rub out the quotes that the ZX81 puts on the bottom line when string input is expected. The easiest way of doing this is to press EDIT (SHIFT/1), which always deletes any input data being typed.

5.10 Using the 16K RAM Pack

Most of the programs in this book were created and tested initially with the 16K RAM pack fitted, then made suitable for the 1K ZX81.

Unfortunately, a program that is created when the RAM pack is fitted and subsequently SAVEd, cannot be re-loaded into the 1K ZX81 without some effort. This is because the display file is set to 24 lines of blank lines once the ZX81 realises that more than 3.75K of memory is available, which is too large to fit into the basic 1K ZX81.

If you want to run a program later on the basic unit, then before you save the program, enter the following direct commands:-

```
POKE 16388,0
POKE 16389,68
CLS
```

The first two commands set the RAMTOP system variable back to the decimal equivalent 17408, while the CLS command ensures that the display file is contracted to a minimum size.

Obviously, your program must be capable of fitting into the 1K ZX81 or this will all be in vain.

5.11 Arithmetic Accuracy

In common with other computers, the ZX81 is incapable of holding all numbers accurately.

Humans use arithmetic to the base 10, arising from the fact that we can count items easily on fingers. However certain values can never be exactly represented by a decimal number, for example PI or SQR(2) - these are called irrational numbers.

The computer suffers the same problem, although the numbers that it cannot accurately represent are slightly different, since it uses arithmetic to the base 2.

This leads to some interesting, but sometimes disturbing, errors in calculations as the results (typically of division) can leave fractional components which are inaccurate. The PRINT command rounds out these small errors, but a few can still creep through into some of your calculations.

Try this:-

```
LET V=100*0.15
PRINT V,INT V
```

You will expect V to contain the value 15 - which it does - and INT V to also give 15, since the INT function rounds down to an integer value. But in this case, INT V gives 14.

You would probably expect to see the error by then writing:-

```
PRINT V-INT V
```

...but to no avail, as the PRINT command removes these errors and prints the value 1.

In this particular case, the problem can be solved quite easily by writing instead:-

```
LET V=100*15/100
```

...which forces greater accuracy on the division since it is then dealing with larger values.

If you intend to write financial programs, statistical or mathematical programs, this can cause you a few headaches. Each occurrence of these rounding errors needs to be treated individually, making it hard to give you a general solution here.

One piece of advice, however, is to ensure that all multiplication is performed before division wherever possible, hopefully ensuring that the division will give a more accurate result. This is a method that you sometimes need to adopt on a calculator in order to avoid losing accuracy - the ZX81 is not really that much different in this respect.

*** Stocking Filler ***

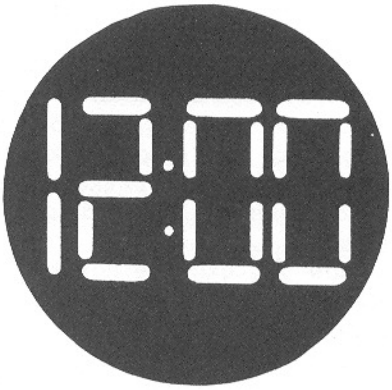
Definitely suitable for : 1K RAM

Yet another page-filling program. This one just draws a pretty pattern.

```
10 FOR X=0 TO 9999
20 SCROLL
30 LET Z=SIN (X*PI/10)*15
40 PRINT AT 21,16-Z;"*";AT 21,16+Z;"*"
50 NEXT X
```

Digital Clock

Suitable for : 1K RAM



This program avoids the pitfalls that are found in the clock program in the Sinclair ZX81 book. The main problem with that particular program is that it cannot be "tuned" accurately, since the PAUSE 42 statement can only ever be altered by units of 1/50th of a second. Although dummy statements can be included in the program in an attempt to add a small time difference, it will never be exactly a multiple of 1/50th second, therefore the clock will never run very true.

After all that, you're probably expecting an atomic clock! Well, not quite, but certainly this version is more accurate than the Sinclair clock. The disadvantage is that the clock must run in SLOW mode - those ZX80 8K ROM users will not be able to use this clock (sorry). Do not despair, though, as the technique involved can be put to good use in other programs which do not rely on a constant display.

The method revolves around the use of system variable FRAMES, which is decremented for each frame sent to the television - 50 times per second. When the ZX81 is running in slow mode, the screen is constantly refreshed, therefore the FRAMES field is always "ticking" fifty times per second.

Since this "ticking" speed is really too fast to monitor in a BASIC program, I have used the most significant half of the FRAMES field (address 16437), which alters every time 256 frames have been sent to the screen. As 50 frames are sent per second, this corresponds to 256 * 0.02 seconds, or 5.12 seconds.

So here we have a "built-in" clock. If a program monitors the contents of address 16437, then whenever its value alters, 5.12 seconds have passed. It does not make any difference that the program takes one or two seconds to process any other work, since in slow mode, the clock is always "ticking". As long as your program takes less than 5.12 seconds to keep checking the clock, it will keep accurate time.

This Digital Clock program uses the ideas above to constantly display the time. It does not display the "seconds" field each second, as the internal "clock" only ticks every 5.12 seconds, so instead, it shows the time accurate to the nearest second every 5.12 seconds. Occasionally, you will notice the time jump by six seconds rather than five. This is when the odd 0.12 seconds eventually build up to become significant (i.e. after 9 "ticks").

Now you can see how tuning this clock becomes accurate. Since the program is no longer dependent upon the PAUSE command, any value can be added in whenever a "tick" is detected. Although the true value of a tick should be 5.12 seconds, you will find that the clock probably doesn't quite run at the right speed. Now you can alter the amount added (line 5 below) by any small amount that you wish.

Run the clock for one hour (as exactly as possible), then calculate the number of ticks that occurred during that period. You may prefer to modify the program to do this for you. Then calculate the length that each tick should take (3600/n) and enter this value into line 5 below. On my own ZX81, I use a tick length of 5 seconds.

The program "16K Fruit Machine" gives a good example of a program which uses the clock for timing the length of a game.

Note that this program occupies almost all of the memory on a 1K ZX81.

Tape name: "CLOCK"

```
  1 REM DIGITAL CLOCK
  5 LET TICK LENGTH=5.12               (see text above
 10 PRINT "ENTER TIME (HHMM)"          (get starting time
 20 INPUT T$
 30 LET H=VAL T$(1 TO 2)               (take out hours
 40 LET M=VAL T$(3 TO 4)               (...and minutes
  50 LET S=(-TICK LENGTH)                  (first time through
  60 LET T1=-1                             ("tick" monitor
  70 POKE 16436,255                        (start clock ticking
  80 CLS
  90 PRINT AT 10,10;"***********";
           AT 12,10;"***********"          (pretty display
 100 LET T2=PEEK 16437                     (monitor FRAMES
 110 LET TICK=(T1<>T2)                     (see if tick occurred
 120 IF TICK THEN GOSUB 1000               (yes?
 130 LET T1=T2                             (update tick monitor
 140 GOTO 100                              (keep trying
1000 LET S=S+TICK LENGTH                   (increment time
1010 IF S<60 THEN GOTO 1100                (has a minute passed?
1020 LET M=M+1                             (update minutes
1030 LET S=S-60                            (reset seconds
1040 IF M<60 THEN GOTO 1100                (has an hour passed?
1050 LET H=H+1                             (update hours
1060 LET M=M-60                            (reset minutes
1070 IF H>23 THEN LET H=0                  (has a day passed?
1100 PRINT AT 11,12;H;":";M;":";INT S;"   "     (print clock
1110 RETURN
```

Artist

Suitable for : 1K RAM

This tiny program turns you into Picasso! Well, perhaps that's a bit too strong a comparison, but it does allow you to draw pictures on the screen.

The program asks for a series of "brush strokes", and each is a five-character string. The first two represent the line number, the next two are the column number, and the last character is the character to be "painted" at that position on the screen. As an example, suppose you want to draw the character "*" at line 18, column 5. You would enter "1805*" when asked.

Obviously, if you make a mistake, you can correct it by drawing a "space" at the same position.

Tape name: "ARTIST"

```
10 INPUT C$                                 (get "brush stroke"
20 IF LEN C$<>5 THEN GOTO 10                (check it's O.K.
30 PRINT AT VAL C$(1 TO 2),VAL C$(3 TO 4);C$(5);
                                            ("paint" it
40 GOTO 10                                  (keep going...
```

Here's an example of a few "brush strokes" to give you the idea (the :: characters are obtained by GRAPHICS/SHIFT/A) :-

```
0920/        1314-        1224::
0921I        1315-        1225::
1019/        1316-        1226::
1021I        1317-        1227::
1114-        1318-        1228::
1115-        1319-        1229::
1116-        1320-        1230::
```

```
1117-        1321-        1121I
1118-        1311-        1221I
1213/        1312-
1214.        1222::
1313-        1223::
```

Those of you with the 16K RAM pack may like to try this alternative program. It additionally lets you save your pictures on tape so that they can be re-drawn whenever you fancy!

Run the program in the usual way, entering "brush strokes" as in the smaller version (the example picture above will work just as well with this version). When you are happy with your picture, enter "SAVE" instead. The program will ask you for the picture name and will then save the program on tape under this name (make sure the cassette is running before you enter the name!).

When you load the program again, it will automatically draw your picture for you.

At any time, you may enter "QUIT" to stop the program.

Tape name: "ARTIST16"

```
  1 REM 16K ARTIST
  2 REM **********
 10 DIM P$(22,32)                        (copy of picture
 20 REM DRAW PICTURE
 30 CLS
 40 FOR Y=1 TO 22
 50 PRINT P$(Y);
 60 NEXT Y
100 REM GET BRUSH STROKES
110 INPUT C$
120 IF C$="SAVE" THEN GOTO 1000
130 IF C$="QUIT" THEN GOTO 9999
140 IF LEN C$<>5 THEN GOTO 100           (check legality
150 LET Y=VAL C$(1 TO 2)                 (line number
160 LET X=VAL C$(3 TO 4)                 (column number
170 IF INT (Y/22)<>0 OR INT (X/44)<>0 THEN GOTO 100
180 PRINT AT Y,X;C$(5);                  ("paint" character
 190 LET P$(Y+1,X+1)=C$(5)          (also store in array
 200 GOTO 100                       (keep going
1000 REM SAVE PICTURE
1010 PRINT AT 21,0;"PICTURE NAME?"
1020 INPUT N$
1030 IF LEN N$=0 THEN GOTO 1000     (don't want error F
1040 SAVE N$
1050 GOTO 20                        (draw the picture again
9999 STOP
```

***** Stocking Filler *****

Suitable for : 1K RAM

The ZX81 thinks of a number between 1 and 99 and you have to guess it before the ZX81 gets bored giving you clues.

```
  10 LET N=INT (RND*99)+1
  20 SCROLL
  30 PRINT "I AM THINKING OF A NUMBER..."
  40 FOR G=1 TO 5+INT (RND*3)
  50 SCROLL
  60 PRINT "ENTER GUESS NO. ";G;": ";
  70 INPUT Y
  80 PRINT Y
  90 IF Y=N THEN GOTO 200
 100 SCROLL
 110 PRINT "NOPE - TOO ";
 120 IF Y>N THEN PRINT "HIGH."
 130 IF Y<N THEN PRINT "LOW."
 140 NEXT G
 150 SCROLL
 160 PRINT "FAILED. THE NUMBER WAS ";N
 170 GOTO 9999
 200 SCROLL
 210 PRINT "CORRECT. PRETTY GOOD..."
9999 STOP
```

DECIMAL JUSTIFICATION

One problem with floating-point arithmetic is that of print formatting. Since the ZX81 always gives an unformatted string when a numeric variable is printed, it is difficult to display a screen of results where the numbers are all in neat columns.

6.1 Justifying money values

The problem mentioned above is particularly apparent when a program is using calculations involving money - two decimal places are required even when these are zero. For instance, a VAT calculation program may want to display the results as:-

```
AMOUNT    VAT RATE      VAT

100.00     15.00       15.00
  1.00     15.00        0.15
 25.60     15.00        3.84
......     .....        ....
......     .....        ....
```

If the program were to print these results without any formatting, they would be left justified - that means they would all line up on the left-hand side:-

```
AMOUNT    VAT RATE      VAT

 100       15          15
 1         15          0.15
 26.6      15          3.84
```

Decimal Justification

This subroutine converts numbers into a fixed format and prints them at a specified column number, making it much easier to read a screen of printed results. If you're using a printer, then these results will look so much smarter.

All results are rounded to two decimal places, so a value of 3.245 will be printed as 3.25. Be careful, though, since you may find that fractional components in your numbers make any totals appear incorrect by one penny from time to time. Try to keep values to two decimal places all the time.

Two variables are required to drive this routine:-

V	which contains the value to be printed
C	which contains the display column number at which the right-hand side is to be aligned.

```
9500 REM JUSTIFY V TO C (2 DEC PLACES)    (Tape name: "JUST2"
9510 LET XL=INT (ABS V+.005)*SGN V        (get pounds rounded
9520 LET XP=INT ((ABS (V-XL)*100)+0.5)    (get pence rounded
9530 LET Z$=STR$ XP                       (convert pence
9540 LET Z$=STR$ XL +"."+("0"+Z$)(LEN Z$ TO )
                                          (force in zeros
9550 PRINT TAB (C-LEN Z$+1);Z$;           (print at col C
9560 RETURN                               (all done
```

A small example of this in use:-

```
100 REM VAT CALCULATOR                (tape name: "VATCALC"
110 PRINT "ENTER AMOUNT £££.PP"
120 INPUT V
130 LET S=V                           (save amount for later
140 LET C=18                          (print at column 18
150 GOSUB 9500                        (justify V and print it
160 PRINT                             (new line
170 LET V=V*15/100                    (calculate VAT
180 GOSUB 9500                        (justify & print
190 PRINT                             (new line
200 PRINT TAB 13;"------"             (separator to look nice
210 LET V=V+S                        (value + VAT
220 GOSUB 9500                       (print justified total
230 PAUSE 4E4
240 CLS
250 RUN
```

Add 235 POKE 16437,255 if you have a ZX80.

6.2 Variable Justification

This is really an extension of the previous routine, but it allows you (additionally) to specify the number of decimal places that are required in the printed output.

As well as variables V and C, you must set variable N to the number of decimal places required in the display.

In this subroutine, rounding is not automatic as there are many cases where the ZX81 will not give accurate results of division (see "Hints 'n' Tips"). You should ensure that variable V is rounded to the required degree of accuracy prior to calling the subroutine.

Tape name: "JUSTN"

```
9500 REM JUSTIFY V TO C WITH N PLACES
9510 LET Z$=""
9515 FOR Z=1 TO N
9520 LET Z$=Z$+"0"
9525 NEXT Z
9530 LET XL=INT ABS V*SGN V
9535 LET XP=INT (ABS (V-XL)*10**N)
9540 LET Z$=STR$ XL+"."+(Z$(1 TO N-LEN STR$ XP)+
          (STR$ XP+Z$))(1 TO N)
9545 PRINT TAB (C-LEN Z$+1);Z$;
9550 RETURN
```

This routine could be used in the VAT calculation given earlier with only one alteration to the original program. A new line number should be added to set the number of decimal places to 2:-

```
        105 LET N=2
```

Obviously, this routine is slightly slower than the previous, as it involves the ** operation to give the decimal fractions, but it does give much greater flexibility. If you only need two places of decimals, then stick to the first version, otherwise make use of this one. Variable N can be set once and left alone, or can be altered to different values throughout the program as necessary.

6.3 Print "Using" Routine

Although the ZX81 BASIC conforms quite well to the Microsoft Basic that has become a fairly-well adopted standard on microcomputers, there are several areas where it does not offer the full features of this Basic.

One such area is the PRINT command, which gives no real formatting power. This routine is an extension of the previous, in that it gives even more flexible numeric formatting power.

Assign string variable U$ to contain a mask of the number as required on printing, e.g. :-

```
        LET U$="9999.99"
```

...will force the number contained in variable V to be printed with two decimal places, and a maximum of four leading digits. The digit "9" represents a position for the resulting number to be placed while the full-stop indicates the relative position of the decimal point. Obviously, since the ZX81 Basic caters for a maximum of nine-and-a-half significant digits, there is no real point in using a mask with more than nine (or possibly ten) mask characters.

This routine will also cater for a mask which does not contain any decimal places, thus making it general purpose for any numeric values (within the constraints of 9.5 digits - beyond this point the ZX81 gives exponential notation, causing the routine one or two headaches!).

If the resulting field requires more printing characters than your mask specifies, then the left-hand characters (i.e. the most significant) are truncated.

Decimal Justification

As with the previous routine, rounding is not performed, so variable V should be set to the required accuracy before entry.

Variable U$ can either be assigned once at the start of a program and retained throughout, or altered to various values as required during a program.

The three variables required are:-

V	the value to be printed
C	the print column of the right-hand edge
U$	the print mask required (see above)

Tape name: "JUSTU"

```
9500 REM PRINT USING U$
9505 LET Z$=""                          (initial "zero" mask
9510 LET XL=0                           (decimal point not found
9515 FOR Z=1 TO LEN U$                  (now search for dec point
9520 IF XL THEN LET Z$=Z$+"0"           (if found, update "zeros"
9525 IF U$(Z)<>"." THEN GOTO 9535       (see if mask has a "."
9530 LET XL=NOT XL                      (set dec. point found
9535 NEXT Z
9540 LET XL=INT ABS V*SGN V             (as before
9545 LET XP=INT (ABS (XL-V)*10**LEN Z$)
9550 IF LEN Z$ THEN GOTO 9565           (check if no dec. places
9555 LET Z$=STR$ XL                     (just print integer
9560 GOTO 9570
9565 LET Z$=STR$ XL+"."+(Z$(1 TO LEN Z$-LEN STR$ XP)+
        STR$ XP+Z$)(1 TO LEN Z$)
9570 IF LEN Z$>LEN U$ THEN LET Z$=Z$(LEN Z$-LEN U$+1 TO )
                                        (truncate Z$ if necessary
9575 PRINT TAB (C-LEN Z$+1);Z$;
9580 RETURN
```

Standard Deviation

Suitable for : 1K RAM

This small program allows you to calculate the standard deviation of a number of data points. Terminate the list by typing END when asked for a value. You are told which value to enter next so that you can keep track of where you have reached.

The use of entering a numeric expression in reply to a request for input data is covered in the section "Hints 'n' Tips".

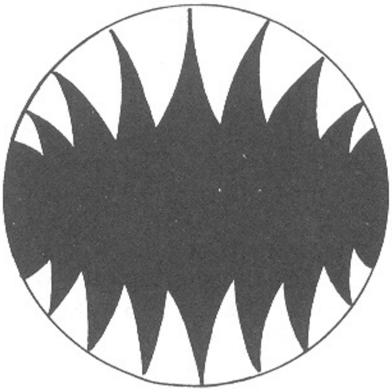
Tape name: "STDDEV"

```
   1 REM STANDARD DEVIATION
  10 LET POINTS=0                          (initial values
  20 LET SQUARES=0
  30 LET SUM=0
  40 LET END=1E38                          (this can be altered
 100 CLS                                   (main program loop
 110 PRINT "ENTER DATA POINT ";POINTS+1    (ask for data
 120 INPUT VALUE                           (input it
 130 IF VALUE=END THEN GOTO 200            (is it the end?
 140 LET POINTS=POINTS+1                   (update no. entered
 150 LET SUM=SUM+VALUE                     (update total so far
 160 LET SQUARES=SQUARES+VALUE**2          (sum of squares
 170 GOTO 100                              (next data point
 200 CLS                                   (print results
 210 PRINT "STANDARD DEVIATION IS ";
 220 PRINT SQR (SQUARES/POINTS-(SUM/POINTS)**2)
9999 STOP
```

Jaws

Suitable for : 16K RAM

Time for an all-action game! You control the shark which is terrorising four innocent swimmers - try to get them all in the minimum number of moves.



You can move the shark by pressing the 5,6,7 and 8 keys (don't use SHIFT), which will move the shark in the appropriate direction of the arrows. Once you have altered course, you do not need to keep pressing the key.

The four swimmers begin at random positions on the screen and each moves in a different direction. Their swimming speed is only one quarter of yours. As they reach the edge of the screen, they will "wrap round" and re-appear on the opposite side.

Tape name: "JAWS"

```
  1 REM JAWS
  2 REM ****
 10 DIM S(6,2)                      (1-4 swimmers, 5-6 you
 20 FOR X=1 TO 4                    (place swimmers in sea
 30 LET S(X,1)=INT (RND*64)         (random row
 40 LET S(X,2)=INT (RND*44)         (  "    column
 50 NEXT X
 60 LET S(5,1)=30                   (place shark
 70 LET S(5,2)=24
 80 LET YD=4                        (initial shark direction
 90 LET T=0                         (number of turns
100 LET N=1                         (next swimmer to move
110 PRINT AT 10,14;"JAWS"
200 REM MAIN LOOP
```

```
 210 LET X=1                                  (set "all eaten" marker
 220 FOR Y=1 TO 4                             (check each swimmer
 230 IF S(Y,1)<>S(5,1) OR S(Y,2)<>S(5,2) THEN GOTO 300
                                              (see if shark at swimmer
 240 PRINT AT 21-INT (S(Y,2)/2),INT (S(Y,1)/2);"MUNCH"
                                              (oh dear...
 250 PAUSE 100
 260 LET S(Y,1)=-1                            (mark swimmer as "eaten"
 300 IF S(Y,1)<>-1 THEN LET X=0               (see if any left alive
 310 NEXT Y
 320 IF NOT X THEN GOTO 500                   (carry on if any left
 330 PRINT AT 20,0;"YOU TOOK ";T;" TURNS"
 340 GOTO 9999
 500 LET D=CODE INKEY$ -32                    (get change of direction
 510 IF D>0 AND D<5 THEN LET YD=D             (check it's valid
 520 LET D=YD                                 (move shark
 530 GOSUB 1000                               (calculate shark directn.
 540 UNPLOT S(6,1),S(6,2)                     (remove shark's tail
 550 LET S(6,1)=S(5,1)                        (keep track of shark tail
 560 LET S(6,2)=S(5,2)
 565 LET S=5                                  (shark subscript
 570 GOSUB 1100                               (draw new head
 580 LET S=N                                  (check next swimmer
 590 IF S(S,1)=-1 THEN GOTO 900               (is he dead?
 600 LET D=N                                  (move swimmer
 610 GOSUB 1000                               (check his direction
 620 UNPLOT S(S,1),S(S,2)                     (remove old position
 630 GOSUB 1100                               (draw new position
 900 LET T=T+1                                (update no. of turns
 910 LET N=N+1                                (update swimmer number
 920 IF N>4 THEN LET N=1                      (only 4 of them
 930 GOTO 200                                 (keep going...
1000 REM CALCULATE DIRECTION
1010 LET DX=((D=1)*-1)+(D=4)                  (DX holds single position
1020 LET DY=((D=2)*-1)+(D=3)                  (DY  "      "       "
1030 RETURN
1100 REM MOVE OBJECT
1110 LET S(S,1)=S(S,1)+DX                     (move object one position
1120 LET S(S,2)=S(S,2)+DY                     (  "     "     "     "
1130 LET S(S,1)=S(S,1)-INT (S(S,1)/64)*64  (check "wrap round"
1140 LET S(S,2)=S(S,2)-INT (S(S,2)/44)*44  (  "      "     "
1150 PLOT S(S,1),S(S,2)                   (draw new position
1160 RETURN
```

If you are using a ZX80, add the following two lines:-

```
505 PAUSE 25
506 POKE 16437,255
```

*** Stocking Filler ***

Definitely suitable for : 1K RAM

A small program to fill up the remainder of a blank page. You'll find some interesting patterns formed at times. Apologies to ZX80 users, but this is strictly for slow mode. You can alter it if you wish, but I think you may find the screen jumping too offputting.

```
10 LET P=0
20 LET X=INT (RND*32)+16
30 LET Y=INT (RND*22)+11
40 IF P THEN PLOT X,Y
50 IF NOT P THEN UNPLOT X,Y
60 LET P=NOT P
70 GOTO 20
```

USING MACHINE CODE

Don't expect me to teach you how! This section shows you ways of entering machine code routines into the ZX81. There are three methods which are all fairly straightforward to use, although each has various advantages and disadvantages. The method you choose will largely depend on the amount of machine code you wish to enter and the way in which it is written.

7.1 Using a REM statement

This method is probably the easiest way of entering a routine that can also be saved with the program on tape.

Your program should include a REM statement as the first line in the program. It should also contain a number of "dummy" characters which will be replaced by your routine.

The first program statement will begin at address 16509 and the address of the first dummy character on this REM statement will be 16514 (there are two bytes for the line number, two for the length of the line and one for the REM token). You can check this by typing :-

```
NEW
10 REM ABCDEF
PRINT CHR$ (PEEK 16514)
```

...which will print "A".

Your program can now POKE machine coded subroutines into this address, overwriting the dummy characters one-by-one.

When you list the program afterwards, the REM statement will look totally weird, as the ZX81 tries to convert all the special codes back into keywords. Don't worry though, as it is only being printed wrongly by the ZX81's listing routine. Here's an example:-

```
  1 REM AAAAA
 10 LET X=16514
 20 POKE X,237
 30 POKE X+1,75
 40 POKE X+2,7
 50 POKE X+3,64
 60 POKE X+4,201
100 PRINT USR X
```

This small program creates a machine code routine which gives the current line number being executed. Whenever USR X is stated, the value of the current line will be returned.

You could use this type of routine to perform "relative goto jumps", since it would be quite legal now to write:-

```
120 GOTO USR X+200
```

...which would continue at line 320, since USR X in this case will give the value 120.

If you wish to save the program with this routine preset, you could delete lines 20 to 60, leaving more room for other program statements. Even if you type RUN, the routine will not be affected in any way, since the routine is held in the program statement and not in a variable.

The disadvantages of this method are:-

1) it clutters your screen up when you list the program,
2) the routine must be held in the first line of the program, a position normally reserved for a REM statement which identifies the program name.

The advantages are:-

1) the routine is safe from RUN and CLEAR,
2) you always know where the routine is located easily, which makes it easy to use directly-addressed jump and call instructions,
3) the routine will be saved with the program, leaving you free to remove all installation statements.

7.2 Using an array

The second method involves creating an array which is to hold the routine within the elements of the array.

The DIM command allocates an array as soon as the command is executed, and so if a DIM command is placed as the first instruction in a program, the array will occupy the initial position in the variables storage section of memory. This address can be found by studying the contents of system variable VARS.

Provided that the array is a single-dimension string array (i.e. a fixed length string), the address of the first character in this string can be obtained by stating:-

```
    LET ADDRESS=PEEK 16400+PEEK 16401*256+6
```

Again, studying page 174 of the Sinclair book will show you that there are six bytes of preamble in a single-dimensioned string array (called "an array of characters" on page 174). This explains why the line above needs "+6" added on the end, as otherwise you will be POKEing into things that you shouldn't!

The example given in the previous section can now be written as:-

```
     1 DIM X$(5)
    10 LET X=PEEK 16400+PEEK 16401*256+6
```

...the rest of the example is then identical, since variable X is used throughout to refer to the location of the routine and is independent of whether it uses an array or a REM statement.

You can, however, use the CHR$ function to assign values with the LET command, so that an alternative solution would be:-

```
  1 DIM X$(5)
 10 LET X=PEEK 16400+PEEK 16401*256+6
 20 LET X$(1)=CHR$ 237
 30 LET X$(2)=CHR$ 75
 40 LET X$(3)=CHR$ 7
 50 LET X$(4)=CHR$ 64
 60 LET X$(5)=CHR$ 201
100 PRINT USR X
```

Again, lines 20-60 can be deleted once the string has been set up properly.

Advantages

1) Your program is not restricted to having a messy REM statement as the first line of program. Provided the DIM command is the first assigned variable, you can put any number of statements in front (e.g. PRINT, FAST, SLOW, etc.),
2) The routine can be manipulated by normal BASIC commands,
3) The routine can be saved with the program, thus all installation statements can be removed.

Disadvantages

1) Any routine must be relocatable (i.e. it can only use relative jump instructions - subroutine call instructions must be simulated by using a stack push and relative jump sequence),
2) All variables must be stored with the program, making it important to avoid RUN and CLEAR and also redimensioning the array,
3) Since there are only 26 distinct string variables, using strings to hold the code could be obstructive.

7.3 Using the top of memory

This method is one of the safest - your machine code is protected from all BASIC commands - even NEW. The only way to destroy the routine is to switch the power off.

Unfortunately, the ZX81 will only SAVE and LOAD the program and variables, thus any routine at the top of memory will not be included.

The system variable RAMTOP holds the address of the top of memory which is set up when the ZX81 is switched on.

You can alter this value at any time by POKEing in a suitable value which lowers this address by the amount of memory you need. You can now POKE your machine code into this address and leave it there all day if necessary.

One problem with this method is that the memory does not become available until NEW is entered - any attempts to POKE into memory beyond RAMTOP until then will simply be ignored.

The approach I have adopted is to use a special "memory loader" program which contains all the necessary machine code to be placed at the top of memory. You must lower the address of RAMTOP by the appropriate number of bytes required for the routine, type NEW, then load and run the loader program, which will move the coding up to the memory at RAMTOP.

If this routine is included at the start of each tape, the program itself can load another program.

Since the loader program will usually serve the single purpose of installing machine code at the top of memory, it can become quite elaborate. The following example program gives an approximate value of the memory available at any time within the ZX81. This can never be extremely accurate as the requirements of a program alter when it is run. It can serve as a general guide, however.

In direct mode, enter:-

```
LET R=PEEK 16388+PEEK 16389*256     (get value of RAMTOP
LET R=R-20                          (reduce it by 20
POKE 16388,R-256*INT (R/256)        (replace it
POKE 16389,INT (R/256)
NEW                                 (don't forget this!!!
```

Now enter this program, save it, then run it:-

```
10 REM MACHINE CODE LOADER          (tape name: "LOADER"
20 LET RAMTOP=PEEK 16388+PEEK 16389*256
30 LET C$="21000039ED5B1C40ED52444DC9"
40 FOR X=1 TO LEN C$-1 STEP 2
50 POKE RAMTOP+INT ((X-1)/2),
     (CODE C$(X)-28)*16+CODE C$(X+1)-28
60 NEXT X
```

This program allows you to enter a machine code routine in hex form. The hex coding is entered into variable C$ in line 30, while lines 40-60 take each pair of characters, convert them into binary, then POKE them into the address at RAMTOP.

Note that although the program is used here to load a routine that calculates the amount of spare memory, it can be used to load any machine code routine merely by replacing line 30.

Once the routine has been installed, it will remain at the top of memory until you switch the ZX81 off.

It can be invoked at any time by specifying:-

```
PRINT USR (PEEK 16388+PEEK 16389*256)
```

...which in this case will give the number of bytes remaining in the ZX81, corresponding to the area marked "spare" in the memory allocation diagram on page 171 of the Sinclair book. The machine code routine you have entered reads:-

```
          Assembler mnemonics     Hexadecimal value

[RAMTOP]: LD   HL,0               21 00 00        ;clear HL
          ADD  HL,SP              39              ;obtain stack ptr
          LD   DE,(STKEND)        ED 5B 1C 40     ;obtain STKEND
          SBC  HL,DE              ED 52           ;give stack-STKEND
          LD   B,H                44              ;answer in BC
          LD   C,L                4D
          RET                     C9              ;return to BASIC
```

Advantages

1) Safe from NEW and CLEAR commands,
2) Extremely long routines can be entered by special programs dedicated to this one task, thereby leaving the maximum amount of memory available for other programs.

Disadvantages

1) The long-winded method needed to get a routine into the top of memory, since a NEW command has to be given somewhere along the route,
2) Unless you are extremely careful with your work, a routine will need to use only relative jumps and calls, otherwise adding extra routines at a later date may be rather tiresome.

7.4 General points

Earlier above, I gave a list of direct commands which reduce the value of the RAMTOP system variable by 20 bytes. The following program is worth adding onto the front of each tape if you intend to keep machine code routines loaded whenever you switch the ZX81 on.

Whenever you want to load a machine code routine (possibly using the loader program above), run this program first to reserve the necessary space:-

```
 1 REM RAMTOP INITIALISATION         (Tape name: "RAMTOP"
 2 REM
 3 REM
10 PRINT "RESERVE HOW MANY BYTES?"
20 INPUT N
30 LET R=PEEK 16388+PEEK 16389*256  (get existing RAMTOP
40 LET R=R-N                         (calculate new value
50 POKE 16388,R-256*INT (R/256)      (put it back again
60 POKE 16389,INT (R/256)            ( "   "   "     "
70 NEW                               (resets memory
```

Now you can run your loader program directly.

If you are interested in machine code in any way, then you may find the next program useful. It allows you to examine portions of memory, specifying addresses in decimal, hex, and will even recognise certain system variables as being the pointer to the start address for a hexadecimal memory dump.

Although it will fit into a 1K RAM ZX81 with subroutine 2000 removed (see below), it is not really so useful in that form.

```
 100 REM MEMORY DISPLAY                  (Tape name: "DUMP16"
 110 GOSUB 9000                          (display rules
 120 LET A=0                             (memory address
 200 GOSUB 1000                          (ask for start address
 210 CLS
 220 FOR Y=0 TO SL                       (number of lines to show
 230 PRINT N+Y*8;TAB 7;                  (starting address
 240 FOR X=0 TO 7                        (print eight bytes
 250 LET Z=PEEK (N+Y*8+X)                (get byte value
 260 PRINT CHR$ (INT (Z/16)+28);         (convert to hex
 270 PRINT CHR$ (Z-(INT (Z/16)*16)+28);
 280 PRINT " ";
 290 NEXT X                              (next byte
 300 PRINT
 310 NEXT Y                              (next line on screen
 320 LET A=N+(SL+1)*8                    (update default address
 330 PRINT
 340 GOTO 200                            (show screen
1000 REM INPUT ADDRESS
1010 PRINT "ENTER ADDRESS (N=CONTINUE)"
1020 INPUT S$
1030 IF NOT LEN S$ THEN GOTO 1020        (ensure something entered
1040 IF S$(LEN S$)="H" THEN GOTO 1100    (check if hex value
1050 GOSUB 2000                          (check for sys. var. name
1060 IF N THEN RETURN                    (return if special name
1070 LET N=VAL S$                        (otherwise it's decimal
1080 RETURN
1100 REM CONVERT HEX TO DECIMAL
```

```
1110 LET N=0
1120 FOR X=1 TO LEN S$-1
1130 LET N=N*16+CODE S$(X)-28
1140 NEXT X
1150 RETURN
2000 REM CHECK FOR SPECIAL ADDRESSES
2010 LET SL=16                          (screen size limit
2020 LET N=0                            (converted address
2030 IF S$="N" THEN LET N=A
2040 IF S$="PROG" THEN LET N=16509
2050 IF S$="VARS" THEN GOTO 2100
2060 IF S$="DFILE" THEN GOTO 2120
2070 IF S$="RAMTOP" THEN GOTO 2140
2080 RETURN
2100 LET N=PEEK 16400+PEEK 16401*256
2110 RETURN
2120 LET N=PEEK 16396+PEEK 16397*256
2130 RETURN
2140 LET N=PEEK 16388+PEEK 16389*256
2150 RETURN
9000 REM INSTRUCTIONS
9010 PRINT TAB 8;"HEX MEMORY DISPLAY"
9020 PRINT
9030 PRINT " ENTER THE START ADDRESS AS :    ",
           " (1) A DECIMAL NUMBER",
           " (2) A HEX VALUE (E.G. 4EA3H)",
           " (3) ""N"" (MEANING ""CONTINUE"")",
           " (4) A SYSTEM VARIABLE NAME -",
           "     PROG,VARS,DFILE,RAMTOP"
9040 PRINT
9050 PRINT "(PRESS A KEY)"
9060 PAUSE 4E4
9070 POKE 16437,255
9080 CLS
9090 RETURN
```

Without the 16K RAM pack, you should replace lines 2000-2150 with the following :-

```
2000 LET SL=8                    (reduce screen size limit
2010 LET N=0
2020 RETURN
```

Do not enter any of the REM statements, also delete lines 1040, 1100-1150, 9000-9090, and line 110. You will be able to enter values in decimal, and you lose the facility of entering system variable names (sorry).

This 1K version is also supplied on the tape as "DUMP".

7.5 Combining routines

This is rather a deluxe machine code routine that allows you to combine several different routines together and specify which particular function is required at different times.

The system variables contain a two-byte spare field at address 16507 which the BASIC system does not affect, although it is cleared whenever NEW is used.

This field is used by the routine to determine which function is required. By POKEing a number into this address, the routine jumps to the appropriate choice. I have included an assembler listing of this routine in Appendix B, but the various features are outlined here.

The routine must begin on a page boundary (i.e. the address must be a multiple of 256). Note that whenever you switch the ZX81 on, the value of RAMTOP will be a multiple of 256, so if you run the initialisation program given above, you can reserve 256 bytes and be sure that the routine will be located on a page boundary.

If you wish to add further routines at a later date, then you should study the listing in the appendix. As supplied, the routine functions are:-

Value POKEd into 16507	Returned value of USR
0	Estimate of spare memory
1	Address of the display file
2	Address of the BASIC variables
3	Address of spare memory above the USR routine itself
4	Current line number.

Most of these functions are rather trivial, although they can be extremely useful by preventing the need for a 2-byte addition statement with PEEKs. Here's an example of the routine in use:-

```
 20 LET R=PEEK 16388+PEEK 16389*256       (get routine address
 30 LET S=16507                           (spare sys. variable
100 POKE S,0                              (spare memory option
110 LET SPARE=USR R                       (get it
120 PRINT "THERE ARE ";SPARE;" BYTES LEFT"    (print it
130 POKE S,2                              (variables address
140 LET V=USR R                           (V now holds address
                                          ( of BASIC variables

....
....
```

Lines 130 and 140 effectively replace:-

```
130 LET V=PEEK 16400+PEEK 16401*256
```

...and save you 18 bytes of program memory.

So what about this super routine? You can enter it by altering line 30 in the "Machine Code Loader" program to the following (shown as you will see it on the screen when you enter it) :-

```
30 LET C$="3A7B40FE05D05F1600211       (Tape name: "SELECTOR"
00060196EE91522272C3121000039ED5
B1C40ED52444DC9ED4B0C40C9ED4B104
0C901003644C9ED4B0740C9"
```

Run the loader program, then try the small test program to see how many spare bytes you have. If you are entering a particularly large program (more so when the 16K RAM pack is attached), it is always good to see whether or not you are heading towards disaster. It takes quite a long time to type a large program, so any indication at an early stage that you may be tight for space is worthwhile.

*** Stocking Filler ***

Definitely suitable for : 1K RAM

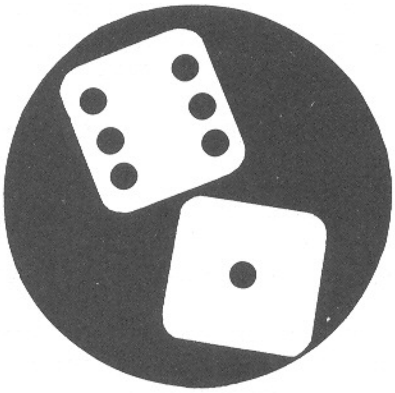
Try to outsmart your opponent (the ZX81). After a short random time, the computer shoots at you. If you press a key before it shoots, you can stop it from firing, but don't be too quick, or it'll shoot anyway!

```
 10 LET R=INT (RND*200)
 20 FOR X=1 TO R+50
 30 IF INKEY$<>"" THEN GOTO 100
 40 NEXT X
 50 PRINT AT 11,14;"**BANG**"        (use inverse characters
 60 GOTO 200
100 IF R-X<10 THEN GOTO 150
110 PRINT AT 9,15;"ITCHY"
120 FOR X=X TO R
130 NEXT X
140 GOTO 50
150 PRINT AT 11,14;"CLICK...",,,,"SAVED"
200 PAUSE 4E4
210 CLS
220 RUN
```

```
ZX80 owners:-
     Alter line 10 to 10 LET R=INT (RND*800)
     Alter line 100 to 100 IF R-X<40 THEN GOTO 150
```

Dice Simulation

Suitable for : 1K RAM



How can you impress your friends? If you are particularly fond of board games, then this program will help. It rolls two dice and shows you the faces. After each roll you only need to press a key to roll again. Who would imagine that £70-worth of computer could be used to simulate dice? It certainly saves you from searching to find them when they've fallen off the table!

I have used a novel technique in writing this program - the ideas were introduced in an earlier section (Efficient Programming), but here you can see some of those points in practice.

The number of numeric literals (e.g. 2 or 8 as opposed to variables) has been kept to an absolute minimum. Since a literal occupies six bytes in a program statement, these can be extremely wasteful when the same literal is used repeatedly. In lines 10-30 of this program, I set up three variables containing the values 2, 4 and 8 (notice how it's done - even this is cheaper than writing LET T=2, LET F=4 and LET E=8 since "LET F=T+T" occupies 11 bytes, whereas "LET F=4" occupies 15).

Wherever a reference to a literal was originally used, I have altered it to use the variables T,F and E instead. In some cases, where the value 12 is needed, I have referred to E+F instead - a saving of 5 bytes each time.

One particularly useful item to note is that the value 1 can be obtained by dividing any variable by itself. Thus T/T equals 1 and also requires 4 bytes less in memory. This may seem ridiculous, but you are sacrificing speed for the sake of memory.

As an exercise, you may like to enter this program, but change all references to E,F and T back into literals. As you enter the program, you'll notice that memory starts to fill between lines 1100 and 1200. The program will not even run - error report 4 is given almost immediately. This shows that the idea is not useless, and you should try to keep an eye on the number of literals that you use in a program.

Tape name: "DICE"

```
   1 REM DICE ROLL
   2 REM
  10 LET T=2                      (set up literals as variables
  20 LET F=T+T
  30 LET E=F+F
  40 RAND
 100 CLS
 110 LET A=E                      (A & B are the screen position
 120 LET B=T+F
 130 GOSUB 1000                   (draw a die face
 140 LET A=E*F                    (place second die
 150 GOSUB 1000                   (draw it
 200 PAUSE 4E4                    (pause forever....
 210 POKE 16437,255
 220 GOTO 100                     (keep doing same thing
1000 LET S=INT (RND*(T+F))+T/T    (get random no. 1-6
1010 FOR X=A TO A+E+E             (draw a box
1020 PLOT X,B
1030 PLOT X,B+E+E
1040 NEXT X
1050 FOR X=B TO B+E+E
1060 PLOT A,X
1070 PLOT A+E+E,X
1080 NEXT X
1100 IF S<>INT (S/T)*T THEN PLOT A+E,B+E   (paint the "spots"
1110 IF S<T THEN RETURN
1120 PLOT A+F,B+E+F
1130 PLOT A+E+F,B+F
1140 IF S<F THEN RETURN
1150 PLOT A+E+F,B+E+F
1160 PLOT A+F,B+F
```

```
1170 IF S<T+F THEN RETURN
1180 PLOT A+F,B+E
1190 PLOT A+E+F,B+E
1200 RETURN
```

*** Stocking Filler ***

Definitely suitable for : 1K RAM

You must take a shot at a target as you run past. Press any key to shoot at the target. Press any key to start a new game afterwards. Run in slow mode (ZX80 users will need to add some PAUSE statements at the appropriate places).

```
 10 LET T=INT (RND*40)+4
 20 PLOT 63,T
100 FOR Y=0 TO 43
110 PLOT 0,Y
120 IF INKEY$<>"" THEN GOTO 200
130 NEXT Y
140 PRINT AT 20,8;"TOO LATE"
150 GOTO 300
200 FOR X=0 TO 63
210 PLOT X,Y
220 NEXT X
230 IF Y=T THEN PRINT AT 21,8;"WELL DONE"
240 IF Y<>T THEN PRINT AT 21,8;"MISSED"
300 PAUSE 4E4
310 CLS
320 RUN
```

ZX80 owners only add:-

```
115 PAUSE 10
305 POKE 16437,255
```

NUMERIC CONVERSION

8.1 Why not use VAL?

Although the ZX81 has an extremely useful **VAL** function, it doesn't quite conform to MicroSoft Basic conventions (this doesn't mean it's bad - just different). The problem is that a program cannot input a string and then use **VAL** to see if the input was numeric, since the act of using **VAL** may give an error B if the string is not a numeric expression.

Why should we need to input a number as a string? There are several reasons, the main one being to make your programs more "robust". This means that invalid information will be spotted before the ZX81 gets a chance to give an error report code.

Try this:-

```
10 PRINT "HOW OLD ARE YOU? ";
20 INPUT A
30 PRINT A
```

Now when you run this, try entering a number like TREE, or CAT or something absurd. You'll quickly get an error of 2/20, or maybe even worse. This obviously isn't much use when you're halfway through playing a game and you enter the wrong reply!

This subroutine lets you convert a string to a numeric variable, and gives an answer of zero if the string was non-numeric. It only caters for positive integer values in the string, although you may have fun converting it to handle both negative numbers and numbers with decimal fractions.

You should notice that the routine will give a zero reply if the string contains the number "0", so it should be used in cases when a zero value is not a legitimate value.

8.2 The Subroutine

To use the subroutine, string variable A$ should contain the string that is to be checked and converted.

On returning, variable X will contain the numeric equivalent of the string (if it was numeric), otherwise it will contain zero.

Tape name: "VAL"

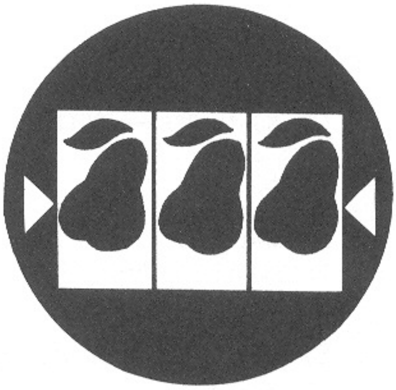
```
9500 REM SET X=VAL(A$)
9510 LET X=0
9520 FOR Y=1 TO LEN A$
9530 IF A$(Y)<"0" OR A$(Y)>"9" THEN GOTO 9570
9540 LET X=X*10+VAL A$(Y)
9550 NEXT Y
9560 RETURN
9570 LET X=0
9580 RETURN
```

Here's the example we used earlier, but altered to use the conversion subroutine. Try to break this. You'll soon see how much more robust it is, and how you have to consciously search for a way to defeat the program!

```
100 PRINT "HOW OLD ARE YOU? ";
110 INPUT A$
120 PRINT A$
130 GOSUB 9500
140 IF X<>0 THEN GOTO 200
150 PRINT A$;"? THATS NOT AN AGE"
160 GOTO 100
200 ....
....
....
```

Fruit Machine

Suitable for : 1K RAM



This program gives you the chance to see how quickly you can spot winning combinations.

The program continually shows a selection of three characters and you may press any key when you see a winning combination being printed. If you are quick enough, your total score will be updated by an amount depending on the combination you have stopped. Winning combinations are:-

* ? ?	Pays 2
* * ?	Pays 10

Any two same pays 5 (except * * ? - see above)
Three of a kind pays 50

You can initially set the speed of the selection - this actually is the time during which the program detects the pressing of a key, where a speed of 50 allows roughly one second in which to stop a winning combination, although you may find this a bit too slow once you have got the idea. If you enter a speed of zero (or even a negative value) you will not be able to stop the program at all (except by using BREAK), since the **INKEY$** function will never be executed!

Press "Q" if you want to quit and see your total score.

There is one point to watch out for - if you stop the program on a non-scoring combination, you lose ten points!

Fruit Machine

Tape name: "FRUIT"

```
  10 LET S=0                                  (initial score
  20 PRINT "SPEED?"
  30 INPUT F
 100 LET Z$=""                                (main loop
 110 FOR X=1 TO 3                             ("roll" wheels
 120 LET Z$=Z$+"*£+-=.$"(INT (RND*7)+1)+" "   (...choose wheel
 130 NEXT X
 140 SCROLL
 150 PRINT Z$;                                (print wheels
 160 FOR X=1 TO F                             (delay for key
 170 IF INKEY$<>"" THEN GOTO 200
 180 NEXT X
 190 GOTO 100                                 (no key - try again
 200 IF INKEY$="Q" THEN GOTO 1000             (see if quitting
 210 LET W=-10                                (false stop score
 220 IF Z$(1)="*" THEN LET W=2                (calculate score
 230 IF Z$(1)=Z$(3) OR Z$(3)=Z$(5) OR Z$(1)=Z$(5) THEN LET W=5
 240 IF Z$(1)=Z$(3) AND Z$(1)="*" THEN LET W=10
 250 IF Z$(1)=Z$(3) AND Z$(1)=Z$(5) THEN LET W=50
 300 PRINT "PAYS ";W                          (print payout
 310 SCROLL
 320 LET S=S+W                                (update total
 330 GOTO 100
1000 PRINT "YOU GOT ";S                       (end of game
```

ZX80 users: Delete line 180, alter line 160 to 160 PAUSE F

If you have a 16K RAM pack, then you may like to add the following lines to improve the play slightly. It allows you to specify how long (in minutes) you wish to play the game. Every time you score a win, the game gets slightly faster, and when you lose, it slows down. The objective is to score the highest possible score within a certain time (say two minutes).

Don't forget that these are additional lines to the previous program, with the exception of line 20 which replaces the original one.

Fruit Machine

Tape name: "FRUIT16"

```
   1 REM 16K FRUIT MACHINE
   2 REM *****************

  20 PRINT "STARTING SPEED?"                 (change message

  35 IF F<1 THEN GOTO 20                     (check speed O.K.

  40 PRINT "HOW LONG? (MINUTES)"             (get time
  50 INPUT N
  60 IF N<1 OR N>10 THEN GOTO 40             (check it's O.K.
  70 POKE 16437,128+N*12                     (start timer

 105 IF PEEK 16437=128 THEN GOTO 2000        (see if time elapsed

 325 LET F=F-(W/10)                          (alter game speed
 327 IF F<1 THEN LET F=1                     (keep eye on speed

1010 STOP                                    (new "time up" code
2000 SCROLL
2010 PRINT "*** TIME UP ***"
2020 SCROLL
2030 GOTO 1000
```

ZX81 Adventure - Create Your Own

Suitable for : 16K RAM

This program allows you to create your own adventure-type games on the ZX81! A complete game is included for you to see how it is done, and also to give you a bit of fun into the bargain.

If you are already acquainted with the original Adventure game, then you may prefer to skip the next few paragraphs.

Adventure was originally written to run on computers somewhat larger than a ZX81 (having roughly 256K bytes of memory available - think about that!), and running faster as well. The game does not vary from one run to another, as the layout of the game remains fixed. Although this may seem a bit limited, the original version is so large that you become bored with playing before you find out all there is to know.

So how is it played? The game takes place in a network of caves, and the object of the game is to gather all the treasure that is dotted around in these caves. Some of the treasure cannot be taken straight away - for example, there is a pearl stuck inside a giant clam, and part of the fun of the game is to discover a way of getting the clam open.

At each turn, you are given a description of your current location. Then the game waits for you to type in up to two keywords. These keywords may ask for you to be moved in a certain direction (for example: GO SOUTH or WALK EAST or CLIMB UP) or to manipulate an object which is in sight - TAKE GOLD or TAKE JEWELS etc.

The objects are not necessarily treasures; they may seem useless at times, but nearly always have some use throughout the game to enable you to get past an obstacle.

In order to give the game some "spice", dwarves pop up at random and throw knives at you (if one hits you, you are killed), although you can retaliate with an axe or some other weapon.

Since the original game is too vast to run on a ZX81, I have borrowed several ideas from different sources to bring a version which can be tailored to run many different games - you will be quite capable of producing your own and swapping them with friends. The game comes in three portions:-

Adventure Master program	which controls the flow of the game and checks that all commands are suitably obeyed
Adventure Loader	which allows you to set up games of your own
Adventure Information	this makes each game unique - it contains the room descriptions, keyword actions, objects etc.

Steps in creating your Adventure

If this is your first encounter with Adventure-type games, I would suggest that you start by entering the games in this section, playing them, then returning to this text when you have a better idea of the capabilities of the game.

The steps involved in creating a game are as follows:-

1. Enter the Master program and the Loader routine together. Save this combination on tape, as it becomes the basis for all your own games.

2. Enter the text messages and room descriptions (use either "The City of Alzan" or the mini-test game found later on).

3. Type RUN 9000 to start the Loader program. Enter all the arrays and variables as given in the appropriate game. The Loader program will automatically save the program for you.

4. Start the game by GOTO 10 (not normally required, since the Loader program ensures that the game starts once loaded from tape).

How the Master program works

The following discussion may seem involved, but if you follow the Test Adventure carefully (I have annotated it fully), you should quickly grasp the way that the Master program works.

Referring to the three portions of the game mentioned above, the Information section requires several items in order to make a complete game. You may like to look at the Test Adventure while you see how the Information section is built up. The items required are:-

1. Room descriptions

   Each room has a unique number associated with it in the range 1 to (max. no. of rooms), and the Master program causes a subroutine call to line 8000+(room*10) so that the room description can be printed. This means that the description for room number 1 should start at line 8010, and a RETURN statement should be included after the description is printed. Room 2 description will be at line 8020, room 3 at 8030, and so on.

2. Text messages

   These are messages which are printed by the Master program on request (see later). The reason for printing these messages can be extremely varied - for example, if the keywords "LIGHT LAMP" are entered, and the lamp is already lit, then you may want to print a message saying so. As for rooms, each message has a unique number starting from 1, and message number 1 should be placed at line number 7010 (followed by a RETURN statement). Message 2 should be placed at line 7020, and so on.

3. Objects

Objects can be used by the player throughout the game - either carried or just manipulated in some way. Each object needs an entry in two arrays - O() which tells the Master program in which room the object is (initially) located, and O$() which contains the text describing the object. This description is 16 characters in length (although you could alter this if required for a particular game). The simple variable O contains the total number of objects. An object which does not initially exist must be given a room number of zero.

4. Vocabulary

This table gives a list of the keywords that your game will recognise as words to be acted upon (in some way). With each word is a two-digit number that this keyword is translated into, so that two different keywords can be translated into the same action (e.g. LEAVE and EXIT would both be given the same two-digit code as they both have the same interpretation). The first twelve keywords are reserved for direction commands (NORTH, SOUTH, UP, DOWN etc), but this is not totally inflexible. These keywords are held in array V$(), and each entry is 6 characters long. The first four are the keyword (only the first four characters of the keyword are matched) and the last two are the two-digit "translated code" number. Note that the code number must have a leading zero for codes less than 10. Variable V should contain the total number of keywords in the array V$().

5. Room connection table

Array M$() contains a table of the "tunnels" connecting each room. For every room, there is an entry showing the number (and direction) of tunnels leading away from the room. Each tunnel requires four characters - two to give the keyword code corresponding to the direction of this tunnel, and two representing the room number that the tunnel leads to (with leading zeros if necessary). A keyword code of "00" indicates the end of the list for this room. As an example, room 1 would be held in M$(1) and might look like:-

M$(1) "0127061300"

This indicates that keyword code 01 will cause the Master program to continue at room 27, while keyword code 06 will cause the Master to continue at room 13. The final 00 signifies the end of the list.

Variable R should contain the total number of rooms. Array M$ allows for 32 characters per room maximum - this will be adequate for almost any set-up.

6. Action table

The action table gives a list of actions to be performed when certain keywords (or combinations of keywords) are entered. Note that direction movements are catered for by the Room Connection table above. Each entry in the Action table gives:-

Keyword 1 code (or 00 if any word is sufficient)
Keyword 2 code (or 00 if any word is sufficient)
Further conditions

Actions to be performed if all conditions are met. A small example at this stage might be :- "If the keywords "LIGHT LAMP" are entered and the lamp is already alight, then display message 5".

A full list of available conditions and actions can be found below, but the overall format of each entry is:-

AABBC01C02.A01A02A03.

...where AA and BB represent the keyword codes for two keywords, C01 C02 represent additional conditions (you may have more than two) followed by a full-stop which indicates the end of conditions. Next follows the appropriate action codes (A01 A02 and A03 - again, you may have more if you wish) also terminated by a full-stop. A complete example is given after the tables below.

Array A$() contains the action table, and variable A must contain the total number of items in the array.

The limit of each entry in array A$() is 31 characters. If this should be insufficient, you can get round the problem by including one action that sets a temporary marker. Further entries can be added into the array C$() which test for this marker and continue the required effects, then unset the temporary marker. This idea is used in "The City of Alzan".

7. Conditional table.

This table is almost identical to the Action table, but does not have any associated keywords. This table is scanned before each command is entered, and allows you to cater for some special circumstances, such as having a dwarf pop up in front of the player, or displaying different messages dependant on certain conditions. Array C$() contains the conditions and variable C contains the total number of items in the array. The format of each entry is:-

C01C02.A01A02.

...where C01 C02 A01 etc are defined as for the Action table. Notice the full-stops that terminate both the conditions and actions.

The various conditions you may use consist of three-character codes. The first is a letter which signifies the desired condition, the last two are a two-digit parameter associated with the condition (shown as nn below):-

Condition code	Test made
A	nn is the current room number
B	Object nn is here (or being carried)
C	Object nn is not here (or being carried)
D	Object nn is being carried
E	Marker nn is set
F	Marker nn is not set
G	Countdown nn has reached value 1
H	Random number from 1-99 is less than nn

Action code	Action performed
A	Print list of objects carried
B	Carry object nn
C	Put down object nn
D	Display text message nn - causes GOSUB to line 7000+(nn*10)
E	Set marker nn
F	Unset marker nn
G	Set countdown nn to the value mm
H	Swap objects nn and nn+1 in the object table
I	Set object nn into current room number
J	Set object nn room number to 00
K	Set current room number to nn (i.e. forced move to room nn)
L	Print "OKAY" and await a new command
M	Await a new command
N	Await a new command, but the Conditional table is not scanned first
O	Describe current room then await new command
P	Abandon the game (player is asked "ARE YOU SURE?" and if the answer is Y then the game is abandoned)
Q	Stop the game

Here is an example of these in use:-

The keyword "TAKE" has (say) the keycode 15, and "GOLD" has the keycode 22. The object "gold" is number 3.

Suitable entries into the Action table might be:-

```
1522B03.B03L.
```

This means:-

Keywords 15 and 22 must be typed (TAKE GOLD), and the condition B03 must also be true (i.e. object 03 must be here or being carried). If this is true, then actions B03 and L are obeyed - object 03 is now carried, and then the message "OKAY" is printed and a new command is input.

There are 10 "markers" that can be set/unset and tested (see the above tables). All markers are initially unset. Markers 1 to 3 have a special significance to the Master program, but all others can be used as required. The special ones are:-

1 Indicates the total number of objects being carried
2 Tells the Master program whether the room is a "dark" room or "light" room - i.e. whether a lamp is required in order to see.
3 Tells the Master program whether a lamp is off or on.

If the room is a "dark" room, and the lamp is off, the Master program prints a message "IT IS DARK. BETTER GET SOME LIGHT OR YOU MAY BE IN TROUBLE".

Marker 2 should be unset when the player is above ground, and set once he goes underground.

There are also 5 countdown markers that are for general use. They can be set to any desired two-digit value by using Action code G. This code requires two parameters - the countdown number and the value it is to be given. E.g. G0105 would set countdown 1 to the value 5. Countdowns 1 to 4 are automatically reduced under certain conditions by the Master program:-

1 Reduced each time a command is entered.
2 Reduced each turn when marker 2 is set (i.e. when the player is in "dark" rooms).
3 Reduced each turn when marker 2 is set but marker 3 is not - i.e. when the player has not got a lamp on but it is dark. This lets you drop the player in a pit after (say) three moves in total darkness.
4 Reduced each time a command is entered (as for countdown number 1).

Condition 7 lets you test if a countdown marker has reached the value 1, so that you can perform some actions once a limit has been reached.

You should note the following points:-

1. A maximum of 5 objects can be carried at any one time (line 4100 in the Master program). Further objects can only be picked up if another object is dropped first.

2. The Master program checks that you are not already carrying an object when it obeys action B, and that you <u>are</u> carrying the object when it obeys action C. This saves quite a considerable number of items in the Action table.

At the end of the program listing, you will find a small "test" adventure of four rooms, which will allow you to see what is going on and also see if your new program works.

You may like to send a copy of your own Adventure games to us at the address given at the front of this book - if we receive enough good ones to publish, we will pay for all those included.

```
  1 REM ZX81 ADVENTURE MASTER          Tape name: "ADVENT"
  2 REM ********************
  3 PRINT "DO NOT USE ""RUN""."        (see pages 95-96
 10 DIM S(10)                          (switch array
 20 DIM C(5)                           (countdown array
  30 LET ROOM=1                           (initial room no.
  40 DIM P$(2,2)                          (keywords 1 & 2
  50 DIM O(O)                             (objects array - type the
                                          (letter "O" and not "Ø".
  60 FOR X=1 TO O                         (set up objects initially
  70 LET O(X)=Q(X)
  80 NEXT X
 100 IF NOT S(2) THEN GOTO 200            (test if darkness
 110 IF C(2) THEN LET C(2)=C(2)-1         (darkness countdown
 120 IF S(3) THEN GOTO 200                (see if lamp on
 130 PRINT "IT IS DARK - BETTER GET SOME",
          "LIGHT OR YOU MAY BE IN TROUBLE."
 140 IF C(3) THEN LET C(3)=C(3)-1         (no lamp countdown
 150 GOTO 1000                            (wait for a command
 200 REM DESCRIBE ROOM
 210 PRINT
 220 GOSUB 8000+ROOM*10                   (print room description
 300 LET F=0                              (reset flag
 310 FOR X=1 TO O                         (print any objects here
 320 IF O(X)<>ROOM THEN GOTO 500
 330 IF F THEN GOTO 400
 340 PRINT ,,"THERE IS ALSO:"
 350 LET F=1
 400 PRINT "    ";O$(X)
 500 NEXT X
1000 REM ACCEPT COMMAND                   (await a command
1010 LET T=1                              (first check automatics
1020 GOTO 2000
1100 IF C(1) THEN LET C(1)=C(1)-1         (countdown every command
1110 IF C(4) THEN LET C(4)=C(4)-1         (countdown every command
1120 PRINT ,,">"                          (prompt - use inverse
1130 INPUT Y$                             (input command
1140 CLS
1150 LET Y=0                              (command scan
1160 PRINT ">";Y$                         (print command at top
1170 LET P$(2)="00"
1200 FOR W=1 TO 2                         (get (up to) two keywords
1210 GOSUB 6000
1220 IF Y>=LEN Y$ THEN GOTO 1300          (check if all scanned
1230 IF P$(W)="00" THEN GOTO 1210        (was the keyword found?
1240 NEXT W                              (next keyword
1300 IF P$(1)<>"00" THEN GOTO 1600       (was at least one word?
1310 PRINT " PARDON?"                    (printed if nothing found
1320 GOTO 100                            (try again
1600 REM CHECK FOR MOVEMENT
1610 LET Z=1                             (now scan movement table
1620 LET T$=M$(ROOM)(Z TO Z+1)           (get matching keyword
1630 IF T$="00" THEN GOTO 1900           (check if end of entry
1640 IF T$<>P$(1) THEN GOTO 1700         (see if it matches word 1
1650 LET ROOM=VAL (M$(ROOM)(Z+2 TO Z+3))
1660 GOTO 100                            (continue in new room
1700 LET Z=Z+4                           (try next match
1710 GOTO 1620
1900 LET T=0                             (set "Action table" flag
1910 LET MATCH=0                         (no match found yet
2000 REM CHECK FOR CONDITIONALS
2010 LET CP=0                            (table subscript number
2100 LET CP=CP+1
2110 IF NOT T THEN GOTO 2300             (see if scanning Action
2120 LET E$=C$(CP)                       (get from Conditionals
2130 GOTO 2600
2300 IF CP<=A THEN GOTO 2400             (have all been scanned?
2310 IF MATCH THEN GOTO 1000             (has a match been found?
2320 PRINT "YOU CANT";                   (print message
2330 IF VAL (P$(1))<13 THEN PRINT " GO THAT WAY";
2340 PRINT "."
2350 GOTO 100                            (try again
2400 IF A$(CP)(1 TO 2)<>P$(1) THEN GOTO 2100
                                         (check if matches key 1
2410 LET Y$=A$(CP)(3 TO 4)               (get keycode 2
2420 IF Y$<>"00" AND Y$<>P$(2) THEN GOTO 2100
2430 LET E$=A$(CP)(5 TO )                (get conditions/actions
2600 REM CONDITIONS
2610 LET E=1                             (now scan further conds.
2700 IF E$(E)="." THEN GOTO 3000         (full-stop ends conds.
2710 LET TYPE=CODE (E$(E))-38            (get condition code
2720 LET N=VAL (E$(E+1 TO E+2))          (get parameter
2800 GOSUB 2900+TYPE*10                  (evaluate if true/false
2810 IF NOT OK THEN GOTO 2100
2820 LET E=E+3                              (try next condition
2830 GOTO 2700
2900 LET OK=(N=ROOM)                        (condition A - see
2905 RETURN                                 ( text
2910 LET OK=(O(N)=ROOM OR O(N)<0)           (condition B
2915 RETURN
2920 LET OK=(O(N)<>ROOM AND O(N)>=0)        (condition C
2925 RETURN
2930 LET OK=(O(N)<0)                        (condition D
2935 RETURN
2940 LET OK=S(N)                            (condition E
2945 RETURN
2950 LET OK=(NOT S(N))                      (condition F
2955 RETURN
2960 LET OK=(C(N)=1)                        (condition G
2965 RETURN
2970 LET OK=((INT (RND*100)+1)<=N)          (condition H
2975 RETURN
3000 REM ACTIONS
3010 LET MATCH=1                            (now perform actions
3020 LET E=E+1
3100 IF E$(E)="." THEN GOTO 2100            (all done?
3110 LET TYPE=CODE (E$(E))-38               (get action code
3120 IF E$(E+1)<>"." THEN LET N=VAL (E$(E+1 TO E+2))
                                            (get any parameter
3200 LET BREAK=0                            (return line number
3210 GOSUB 4000+TYPE*100                    (perform action
3220 IF BREAK THEN GOTO BREAK               (goto relevant line
3230 LET E=E+3                              (next action
3240 GOTO 3100
4000 PRINT                                  (action A - see table
4010 PRINT "YOU ARE HOLDING:"
4020 LET F=1
4030 FOR X=1 TO O                           (the letter "O" not "Ø"
4040 IF O(X)>=0 THEN GOTO 4070
4050 PRINT "    ";O$(X)
4060 LET F=0
4070 NEXT X
```

```
4080 IF F THEN PRINT "   NOTHING."
4090 LET BREAK=100
4095 RETURN
4100 IF S(1)<5 THEN GOTO 4140          (action B
4110 PRINT "YOU CANNOT CARRY MORE."
4120 LET BREAK=100
4130 RETURN
4140 IF O(N)=-1 THEN GOTO 4180
4150 LET O(N)=-1
4160 LET S(1)=S(1)+1
4170 RETURN
4180 PRINT "YOU ALREADY HAVE IT."
4190 GOTO 4120
4200 IF O(N)=-1 THEN GOTO 4240         (action C
4210 PRINT "YOU DONT HAVE ";O$(N)
4220 LET BREAK=100
4230 RETURN
4240 LET O(N)=ROOM
4250 LET S(1)=S(1)-1
4260 RETURN
4300 PRINT                             (action D
4310 GOSUB 7000+N*10
4320 RETURN
4400 LET S(N)=1                        (action E
4410 RETURN
4500 LET S(N)=0                        (action F
4510 RETURN
4600 LET C(N)=VAL (E$(E+3 TO E+4))     (action G
4610 LET E=E+2
4620 RETURN
4700 LET X=O(N)                        (action H
4710 LET O(N)=O(N+1)
4720 LET O(N+1)=X
4730 RETURN
4800 LET O(N)=ROOM                     (action I
4810 RETURN
4900 IF O(N)<0 THEN LET S(1)=S(1)-1    (action J
4910 LET O(N)=0
4920 RETURN
```

```
5000 LET ROOM=N                         (action K
5010 RETURN
5100 PRINT " OKAY."                     (action L
5200 LET BREAK=1000                     (action M
5210 RETURN
5300 LET BREAK=1100                     (action N
5310 RETURN
5400 LET BREAK=100                      (action O
5410 RETURN
5500 PRINT " ARE YOU SURE? ";           (action P
5510 INPUT W$
5520 PRINT W$
5525 LET BREAK=1100
5530 IF CHR$ CODE W$<>"Y" THEN GOTO 5400
5600 GOTO 9999                          (action Q
6000 REM REMOVE WORD
6010 DIM W$(4)                          (first four letters
6015 LET P$(W)="00"                     (set "not found" reply
6020 GOSUB 6600                         (find first character
6025 IF END THEN RETURN                 (test if end of command
6030 FOR Q=1 TO 4                       (get four letters
6040 LET W$(Q)=Y$(Y)
6050 GOSUB 6500                         (check if word end
6060 IF END THEN GOTO 6100
6070 NEXT Q
6080 GOSUB 6500                         (look for end of word
6090 IF NOT END THEN GOTO 6080
6100 IF W$="    " THEN RETURN           (no word entered
6110 FOR Q=1 TO V                       (scan vocabulary table
6120 IF W$=V$(Q)(3 TO ) THEN GOTO 6200
6130 NEXT Q
6140 RETURN                             (not found in table
6200 LET P$(W)=V$(Q)( TO 2)             (get keyword code number
6210 RETURN
6500 LET Y=Y+1                          (check for end of word
6510 LET END=(Y>LEN Y$)
6520 IF END THEN RETURN
6530 LET END=(Y$(Y)=" ")                (don't forget the space!
6540 RETURN
6600 LET Y=Y+1                          (look for end of word
6610 LET END=(Y>LEN Y$)
6620 IF END THEN RETURN
6630 IF Y$(Y)=" " THEN GOTO 6600      (don't forget the space!
6640 RETURN
7000 REM ACTION MESSAGES
7001 REM MESSAGE NO. 1 CAUSES
7002 REM GOSUB TO LINE 7010
7999 RETURN
8000 REM ROOM DESCRIPTIONS
8001 REM ROOM 1 CAUSES A
8002 REM GOSUB TO LINE 8010
9999 STOP
```

Now that you have the "central manager" portion, you need two further items before you can start running a game. First is a "game loader" routine that initialises all the essential arrays (like the object table, interconnecting room table, vocabulary table, and the automatic and conditional event tables) plus a few other variables. The second item is the game itself - all you have here is something that allows you to quickly create your own adventures.

Two complete mini-adventures are included here for you to load and run yourself, so that you can see how it all fits together (and have a laugh, I hope!), but first, here's the loader routine.

Adventure Loader

This routine should be included with the manager program above, but it can be removed once the various arrays have been properly installed. I would advise you to keep the routine in your game until you have tested it properly - if you miss a few words out of the vocabulary table (or any other table) and you want to re-enter it, then you'll feel mad if you've just deleted the loader routine!

When you run it, it asks how many items are required in each array (like the vocabulary table), then dimensions the array, and inputs the elements one-by-one. It stops after each array has been created, thus giving you an opportunity to check what you've done. If you are re-entering an array that was incorrect, it also gives you a chance to re-input only one array and not all of them.

```
9000 REM GAME ARRAY LOADER
9010 CLS
9020 PRINT "NO. OF OBJECTS?"
9030 INPUT O
9040 DIM Q(O)
9050 DIM O$(O,16)
9080 FOR X=1 TO O                 (type the letter "O" not "Ø".
9090 SCROLL
9100 PRINT "NO. ";X;" ROOM?",
9110 INPUT Q(X)
9120 PRINT Q(X)
9130 SCROLL
9140 PRINT "DESCRIPTION?",
9150 INPUT O$(X)
9160 PRINT O$(X)
9170 NEXT X
9199 STOP                         (use CONT to continue
9200 CLS
9210 PRINT "NO. OF WORDS?"
9220 INPUT V
9230 DIM V$(V,6)
9240 FOR X=1 TO V
9250 SCROLL
9260 INPUT V$(X)
9270 PRINT V$(X)
9280 NEXT X
9299 STOP                   (use CONT to continue
9300 CLS
9310 PRINT "NO. OF ROOMS?"
9320 INPUT R
9330 DIM M$(R,32)
9340 FOR X=1 TO R
9350 SCROLL
9360 INPUT M$(X)
9370 PRINT M$(X)
```

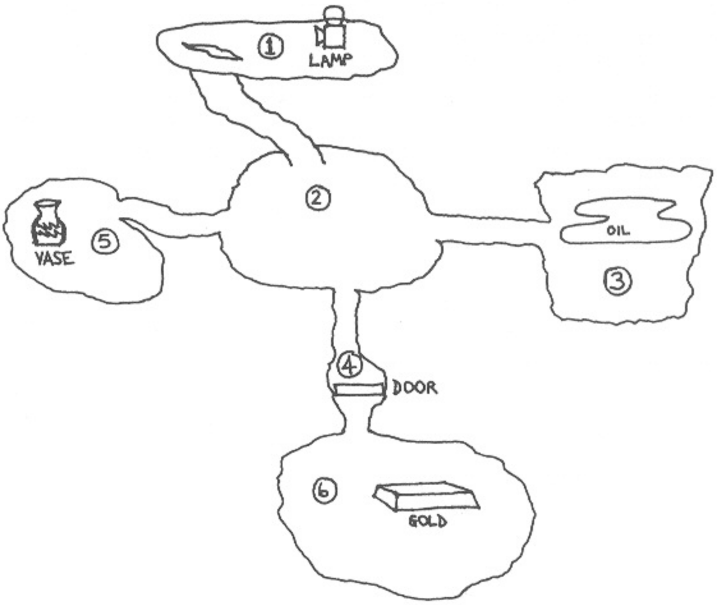
```
9380 NEXT X
9399 STOP                    (use CONT to continue
9400 CLS
9410 PRINT "NO. OF CONDITIONALS?"
9420 INPUT C
9425 LET C=C+1
9430 DIM C$(C,21)
9440 FOR X=1 TO C-1
9450 SCROLL
9460 INPUT C$(X)
9470 PRINT C$(X)
9480 NEXT X
9490 LET C$(C)=".N."
9499 STOP                    (use CONT to continue
9500 CLS
9510 PRINT "NO. OF ACTIONS?"
9520 INPUT A
9530 DIM A$(A,31)
9540 FOR X=1 TO A
9550 SCROLL
9560 INPUT A$(X)
9570 PRINT A$(X)
9580 NEXT X
9599 STOP                    (use CONT to continue
9600 CLS
9610 PRINT "ENTER THE ADVENTURE NAME"
9620 INPUT N$
9630 PRINT ,,"START THE TAPE..."
9640 PAUSE 150
9645 POKE 16437,255
9650 CLS
9660 SAVE N$
9670 GOTO 10
9999 STOP
```

Before you get a real-live game, here follows a "test" Adventure for you to enter. It should give you a good idea as to how the program works, how you can create your own Adventures, and whether all your typing has been accurate.

Test Adventure

This mini-test adventure uses 6 rooms. Room 1 is above ground, and a lamp can be found there. The objective is to get the bar of gold out of the caves back above ground. The gold is hidden in cave 6 behind a rusty door (cave 4), which will not open. Cave 5 contains a vase and cave 6 contains a pool of oil. Obviously, you must fill the vase with oil and then oil the door! Once this has been done, you can open the door and reach the gold.

A map of the cave looks like this:-



Markers 2 and 3 are used, as usual, to represent "dark" rooms and lamp off/on respectively. Notice that when the "lit" lamp (object number 2) is dropped, the lamp is marked as off, which prevents you from lighting the lamp then leaving it somewhere while you wander off.

Marker 5 is used to indicate when the door has been oiled, and marker 6 is set when the door is open.

Enter the text messages and room descriptions, then RUN 9000 to start the Loader routine. The objects, vocabulary, room connections, conditionals and keyword actions are all entered at this stage.

Once you have completed this, SAVE THE PROGRAM!!!

Start the program by GOTO 10 (otherwise you'll destroy the variables). Check that it works according to the rules above and you can be fairly sure that you have entered your Master program without any serious defects.

You should notice the way this is created in order to assist you with producing your own games.

One item of importance is shown between rooms 1 & 2 and also 4 & 6. In the first case, there is no tunnel indicated in the room connection table between rooms 1 and 2. Instead, an entry has been included in the Action table under the appropriate keyword (06). This is because I want to make sure that the "darkness" marker is set on whenever the keyword "DOWN" is given from room number 1.

Similarly, there would be no point in entering a connection between rooms 4 and 6, since the door is supposed to block the path. Consequently, an entry is found in the Action table (03 00 A04 F06....) which checks marker 6 whenever "SOUTH" is entered at room 4.

The rule is:- If you want to place some conditions on the player when he travels from one particular room to another, don't put an entry in the room connection table - use the Action table instead.

Text Messages:-

```
7010 PRINT "THE DOOR IS SHUT FAST"
7015 RETURN
7020 PRINT "THE DOOR IS OPEN"
7025 RETURN
7030 PRINT "IT IS ALREADY ALIGHT"
7035 RETURN
7040 PRINT "WITH A GRUNT YOU MANAGE TO",
           "OPEN THE DOOR."
7045 RETURN
7050 PRINT "IT IS TOO STIFF FOR YOU",
           "TO OPEN."
7055 RETURN
7060 PRINT "YOU DID IT. WELL DONE."
7065 RETURN
7070 PRINT "YOU CANNOT GET PAST THE DOOR."
7075 RETURN
```

Room Descriptions:-

```
8010 PRINT "YOU ARE STANDING BY A POTHOLE."
8015 RETURN
8020 PRINT "THIS IS A VAST CAVERN WITH",
           "PASSAGES LEADING EAST,SOUTH,"
           "AND WEST. A DIM PASSAGE SLOPES"
           "UPWARDS BEHIND YOU."
8025 RETURN
8030 PRINT "THIS CAVE CONTAINS ONLY A POOL",
           "OF OIL."
8035 RETURN
8040 PRINT "HERE IS A GIANT RUSTY DOOR."
8045 RETURN
8050 PRINT "YOU ARE IN THE WESTERN ALCOVE."
8055 RETURN
8060 PRINT "YOU ARE IN THE TREASURE CAVE."
8065 RETURN
```

Test Adventure

Objects:-

Number of objects:- 5

No.	Room number	Description
1	1	A LAMP
2	0	A LIGHTED LAMP
3	5	A MING VASE
4	0	A VASE OF OIL
5	6	A BAR OF GOLD

Vocabulary:-

Number of words:- 25

Each entry below requires a maximum of six characters, the first two being the word number.

```
01N          14DROP
01NORT       15VASE
02E          16GOLD
02EAST       17DOOR
03S          18OPEN
03SOUT       19LAMP
04W          20LIGH
04WEST       21FILL
05U          22OIL
05UP         23INVE
06D          24QUIT
06DOWN       25LOOK
13TAKE
```

Test Adventure

Number of rooms:- 6

Room connection table (the numbers in brackets are for reference only - do not enter them) :-

```
(1)  00
(2)  02030304040500
(3)  040200
(4)  010200
(5)  020200
(6)  010400
```

Number of conditionals:- 3

Conditionals (do not enter the spaces!) :-

```
A04 E06. D02 N.              (room 4 and marker 6 is set -
                             (i.e. the door is open.
A04 F06. D01 N.              (room 4 and M6 is not set ie
                             (the door is shut
A01 D05. D06 Q.              (room 1 carrying object 5 -
                             (got out with the gold - win!
```

Number of keyword actions:- 21

Keyword actions:-

```
13 19 B01. B01 L.            (take lamp - object 01
14 19 B01. C01 L.            (drop lamp
13 19 B02. B02 E03 L.        (take (lit) lamp - object 02
                             ( - also sets lamp marker 3
14 19 B02. C02 F03 L.        (drop lit lamp - unsets lamp
                             ( marker 3
20 00 D01. H01 E03 L.        (light lamp - swaps objects 1
                             ( and 2, also sets lamp mark 3
20 00 B02. D03 M.            (light lamp and object 2 is
                             ( already here - display 03
06 00 A01. E02 K02 O.          (DOWN when at room 1, so set
                               ( "dark" marker 2, continue at
                               ( room number 2
05 00 A02. F02 K01 O.          (UP when at room 2, so unset
                               ( "dark" marker and continue
                               ( at room number 1
13 15 B03. B03 L.              (take vase
14 15 B03. C03 L.              (drop vase
13 16 B05. B05 L.              (take gold
14 16 B05. C05 L.              (drop gold
21 00 B03 A03. H03 L.          (fill vase - must have
                               ( object 3 and be in room 3
                               ( swaps objects 3 & 4
22 00 A04 B04. H03 E05 L.      (oil door - must have object 4
                               ( a full vase and be at room 4
                               ( "empties" bottle & set mark5
18 00 A04 E05. D04 E06 M.       (open door - must be oiled i.e
                               ( marker 5 must be set, and
                               ( must be at room 4. Sets M6.
18 00 A04 F05. D05 M.          (open door when not oiled.
                               ( - displays message 5.
03 00 A04 F06. D07 M.          (SOUTH when marker 6 not set -
                               ( i.e. door not open.
03 00 A04 E06. K06 O.          (SOUTH at door when open [mark
                               ( 6 is set]. Continues at room
                               ( number 6.
23 00 .A.                      (give inventory
24 00 .P.                      (quit
25 00 .O.                      (look to see where we are
```

Testing your Adventures

What do you do if your new Adventure does not work? Here are a few guidelines to help you track down any errors.

From my own experience, the most common problem occurs in the Conditional and Action tables - either specifying incorrect actions, or not entering appropriate items.

You can run the program in either slow or fast mode, but I must point out that it can take quite a few seconds to scan the Action table to match your keywords and so I would recommend fast mode.

If nothing happens for one or two minutes, then suspect a fault in the Conditional table. Press the BREAK key, and inspect any of the following variables by using direct PRINT commands:-

CP	Contains the current Conditional/Action table entry number being processed.
T	indicates whether the Conditional or Action table is being scanned. Zero means Action table, 1 means Conditional table.
E$	contains a copy of the Conditional/Action table entry.
P$(1)	contains the keyword number of the first keyword found in any input command.
P$(2)	contains the keyword number of the second keyword found, or "00" if no second keyword was entered.
ROOM	the current room number.
S(n)	switch n - 0=off, 1=on.
C(n)	countdown n - 0=countdown not in use.
O(n)	room number containing object n. If the object is being carried, this will have the value -1. If the object does not exist, the value will be zero.

Testing Your Adventures

The following arrays/variables are set up by the loader program:-

M$()	room connection table
R	number of rooms
O	number of objects
Q()	object location table, copied into O()
O$()	object descriptions
V	number of words
V$()	vocabulary table
C	number of conditionals
C$()	conditional table
A	number of actions
A$()	action table

A common mistake is to terminate a Conditional table entry with action code M instead of N. Action code M causes the conditional table to be scanned again, and since the same conditionals (probably) still exist, the same program will simply loop indefinitely. Check that all conditionals finish with action code N unless you have good reason to use another (look at the tables in the "Test" Adventure).

For further information on Adventure, read the article by Ken Reed, Practical Computing Vol. 3, Issue 8 (August 1980).

City Of Alzan

Suitable for : 16K RAM

Now for a complete Adventure, based on the "Do-it-yourself" Adventure Master program.

This takes place in a fictitious city named Alzan, which is built on top of the sea cliffs and is inhabited by thieves and cut-throats. Your quest is to find a way out of the city before they grab you, or before the plague takes hold of you. Unfortunately, the city is surrounded by extremely high walls and so you must find a way to scale them.

When you enter the game, it is possible for you to work out how the game evolves and how to win, but this would defeat the pleasure of playing, so try to "switch off" while you are typing.

When this program is fully running, you will have roughly 4150 bytes of memory free. This should give you a guide to the size of Adventures you will be able to write. I would advise the use of the "Memory Left" routine (see "Using Machine Code") while developing your own games.

Tape name: "ALZAN"

Enter the text messages:-

```
7010 PRINT "OH DEAR. YOU MUST HAVE CAUGHT",
           "THE PLAGUE IN THE TOMB. IT",
           "SEEMS THAT YOU HAVE DIED."
7015 RETURN
7020 PRINT TAB 12;"---WHOOSH---"
7022 PRINT "EL GRABBO, THE LOCAL THIEF,",
           "SNATCHES YOUR MONEY AND DIS-",
           "APPEARS INTO THE SEA MIST."
7025 RETURN
7030 PRINT """STOP THIEF""" SHOUTS THE USHER,",
           "BUT YOU MANAGE TO ESCAPE."
7035 RETURN
7040 PRINT "THE COVER IS ALREADY OPEN."
7045 RETURN
7050 PRINT "IT COSTS MORE THAN YOU CAN AFFORD."
7055 RETURN
7060 PRINT "THATLL DO NICELY, SIR"
7065 RETURN
7070 PRINT "THE MANHOLE COVER IS OPEN."
7075 RETURN
7080 PRINT "THE MANHOLE COVER IS SHUT."
7085 RETURN
7090 PRINT "THE SHOPKEEPER IS BIGGER THAN",
           "YOU..."
7095 RETURN
7100 PRINT "YOU WILL NEED A LADDER TO GET",
           "OVER THESE WALLS."
7105 RETURN
7110 PRINT "IT IS ALREADY ON."
7115 RETURN
7120 PRINT "WHAT A STROKE OF GENIUS"
7125 RETURN
7130 PRINT "YOU CATCH THE GUARDS UNAWARE AND";
           "MANAGE TO SNATCH A WAD OF NOTES.";
           "NO-ONE HAS NOTICED (FUNNY LOT,",
           "THESE ALZANS)"
7135 RETURN
7140 PRINT "YOU HAVE TAKEN ALL THERE IS."
7145 RETURN
7150 PRINT "I DONT SEE A TORCH?"
7155 RETURN
7160 PRINT "THE CINEMA IS BOOKED FOR A",
           "PRIVATE FUNCTION."
7165 RETURN

8010 PRINT TAB 8;"WELCOME TO ALZAN",,,
          "YOU MUST SCALE THE WALLS IF",
          "YOU WISH TO ESCAPE FROM THIS",
          "CITY OF THIEVES AND CUT-THROATS."
8015 RETURN
8020 PRINT "YOU ARE IN THE MAIN STREET OUT-",
          "SIDE A HARDWARE SHOP. THE STREET";
          "STRETCHES EAST/WEST AND A SMALL",
          "ALLEY LEADS NORTH BESIDE THE",
          "SHOP."
8025 RETURN
8030 PRINT "YOU ARE INSIDE THE SHOP. THE",
          "SHOPKEEPER LOOKS SHIFTY, BUT HE",
          "HAS MANY FINE GOODS ON DISPLAY."
8035 RETURN
8040 PRINT "YOU ARE IN AN ALLEY BEHIND THE",
          "TALL BUILDINGS. THERE ARE MANY",
          "FULL DUSTBINS UNDER THE FIRE",
          "ESCAPE."
8045 RETURN
8050 PRINT "YOU ARE ON THE FIRE ESCAPE,",
          "WHICH LEADS PAST A DOOR IN THE",
          "BUILDINGS."
8055 RETURN
8060 PRINT "YOU HAVE COME DOWN A SECRET",
          "STAIRCASE INTO THE SHOP."
8065 RETURN
8070 PRINT "YOU ARE ON SOME CATWALKS BETWEEN";
          "THE BUILDINGS."
8075 RETURN
8080 PRINT "THIS IS PART OF THE CITY WALLS.",
          "THERE IS AN UNUSED DOOR IN THE",
          "WALL HERE."
8085 RETURN
8090 PRINT "YOU ARE AT A CROSSROADS."
8095 RETURN
8100 PRINT "HERE IS PART OF THE CITY WALLS.",
          "THE SEA MIST IS QUITE THICK,",
          "MAKING IT HARD TO SEE FAR."
8105 RETURN
```

```
8110 PRINT "YOU PLUNGE FROM THE WALL - RIGHT";
           "DOWN ONTO THE ROCKS BY THE SEA",
           "500FT BELOW. WELL, NEVER MIND,",
           "BETTER LUCK NEXT TIME."
8115 RETURN
8120 PRINT "YOU ARE OUTSIDE THE TOWN BANK."
8125 RETURN
8130 PRINT "INSIDE THE BANK THERE ARE MANY",
           "GUARDS WHO SEEM RATHER BORED."
8135 RETURN
8140 PRINT "YOU HAVE ARRIVED AT A DEAD END,",
           "BUT THERE IS A MANHOLE IN THE",
           "ROAD..."
8145 RETURN
8150 PRINT "YOU ARE IN A SMALL ALCOVE UNDER-";
           "NEATH THE MANHOLE. A PASSAGE",
           "LEADS SOUTH."
8155 RETURN
8160 PRINT "THE PASSAGE LEADS TO AN ANCIENT",
           "TOMB, WHERE MANY SARCOPHAGI LIE",
           "SCATTERED ABOUT."
8165 RETURN
8170 PRINT "THE USHER WILL NOT LET YOU IN AS";
           "THE PROGRAMME HAS STARTED. HE",
           "BLOCKS YOUR PATH WITH HIS TORCH."
8175 RETURN
8180 PRINT "YOU ARE OUTSIDE THE CINEMA.",
           "SOUNDS OF GUNFIRE COME FROM",
           "WITHIN."
8185 RETURN
8190 PRINT TAB 8;"***CONGRATULATIONS***",
           "YOU MADE IT OUTSIDE THE CITY",
           "WALLS. THIS IS INDEED A RARE",
           "OCCASION. WELL DONE."
8195 RETURN
```

Now type RUN 9000 to start the initialisation routine. The arrays should be set as follows:-

City of Alzan

Number of objects : 11

Object room	Description
0	A LIGHTED TORCH
0	A TORCH
3	A LADDER
3	A HAMMER
0	A HAMMER
0	A WAD OF NOTES
0	MANHOLE COVER
15	A BAG OF NAILS
16	A BARCLAYCARD
0	A ROUGH LADDER
4	SOME WOOD

Number of words : 43

```
01N          19HAMM
01NORT       20WAD
02E          20NOTE
02EAST       22BAG
03S          22NAIL
03SOUT       23BARC
04W          05SCAL
04WEST       05CLIM
05U          29OPEN
05UP         29LIFT
06D          30MAKE
06DOWN       30BUIL
13TAKE       31SWIT
14PUT        31LIGH
14DROP       32BUY
15ENTE       33WOOD
15IN         34ROB
16OUT        34STEA
16EXIT       35INVE
16LEAV       36QUIT
17TORC       37LOOK
18LADD
```

City of Alzan

Number of rooms : 19

Room no	Connections
1	00
2	01 04 02 09 04 18 00
3	00
4	02 02 05 05 00
5	06 04 04 07 00
6	00
7	01 08 03 05 00
8	03 07 00
9	01 12 02 10 03 14 04 02 00
10	04 09 00
11	00
12	02 09 04 18 00
13	00
14	01 09 00
15	03 16 00
16	01 15 00
17	00
18	01 12 02 02 00
19	00

Number of conditionals : 9

Conditional table (the spaces are only to make it easier to read - do not enter them) :-

```
A01. K02 O.
A16 H30. G0121.
G01. D01 Q.
B06 H10. D02 J06.
A14 E07. D07 N.
A14 F07. D08 N.
A11. Q.
A19. Q.
A06. K03 O.
```

City of Alzan

Number of actions : 47

Action table (the spaces are to make it easier to read - do not enter them) :-

```
13 17 B01. B01 E03 L.
13 17 A17 C01 C02. I02 B02 D03 K18 E10 O.
32 18 B03. D05 N.
13 19 B05. B05 L.
13 20 B06. B06 L.
29 00 A14 E07. D04 N.
29 00 A14. E07 M.
13 22 B08. B08 L.
13 23 B09. B09 L.
14 17 B01. C01 F03 L.
14 17 B02. C02 L.
14 19 B05. C05 L.
14 20 B06. C06 L.
14 22 B08. C08 L.
14 23 B09. C09 L.
05 00 A10 C10. D10 M.
05 00 A08 C10. D10 M.
05 00 A10. K11 O.
05 00 A08. K19 O.
05 00 A15. F02 K14 O.
06 00 A14. E02 K15 O.
31 00 D02. H01 E03 L.
31 00 B01. D11 N.
32 19 B04 B06. H04 J06 B05 L.
32 19 B04 B09. H04 D06 B05 M.
30 00 B05 B11 B08. D12 I10 J08 J11 M.
13 33 B11. B11 L.
14 33 B11. C11 L.
15 00 A02. K03 O.
15 00 A12. K13 O.
15 00 A18 F10. K17 O.
16 00 A03. K02 O.
16 00 A13. K12 O.
16 00 A17. K18 O.
```

```
15 00 A05. K06 O.
34 00 A03. D09 M.
34 00 A13 E08. D14 M.
34 00 A13. E08 D13 I06 B06 M.
15 00 A18 E10. D16 M.
13 18 B10. B10 L.
14 18 B10. C10 L.
13 18 B03. D09 M.
13 17 B02. B02 L.
35 00 .A.
36 00 .P.
37 00 .O.
50 00 .N.
```

Now use the "save" routine giving the name ALZAN to this adventure. When you next load the program, it will automatically start running, but if you wish to begin again for any reason, use GOTO 1, as the RUN command will clear all variables thus destroying the game.

Have fun!

ZX80 TO ZX81 CONVERSION

There must be literally thousands of programs written for the original 4K ROM ZX80 which can no longer be directly used on the ZX81 (or even the ZX80 with 8K ROM).

This brief appendix gives you a minimal idea of how to convert a program written for the 4K ROM ZX80.

1. All array subscripts for the ZX80 could start at zero, whereas the ZX81 must start at 1. Any program which uses the "zero subscript" must be altered to start at 1. One quick method (not always guaranteed) is to add one to each subscript value that you see used in the program.

2. The ZX80 made use of a special string function TL$, which supplied the following string expression with first character removed. This can be replaced by using the qualifier (2 TO) following the string variable name. For example:-
   ```
        LET A$=TL$(X$)
   ```
   ...will now become...
   ```
        LET A$=X$(2 TO )
   ```

3. The RND function worked in a different fashion - the argument specified the maximum limit of random number value, so that RND(100) gave a random number between 1 and 100. This can be altered as follows:-
   ```
        LET N=RND(X)
   ```
   ...should now be written...
   ```
        LET N=INT (RND*X)+1
   ```

4. All arithmetic involved integer values only, and division results were always rounded down. Unless there is

good reason, alter all divisions to use the INT function on the result.

5. The comma delimiter in PRINT statements moved the print position across to the next quarter of the screen instead of half-way across. You must study the program carefully, but the TAB function may help to move across to the next position. Print zones were placed at positions corresponding to:-
   TAB 8, TAB 16, TAB 24 and TAB 0 (beginning of line).

6. Programs which used PEEK and POKE cannot readily be converted - you must know what effect the commands are having upon the program, and it would take too much space here to include a list of the system variables of the old ZX80.

7. Some of the character code values have altered - particularly the graphic characters - and so you should also be wary of any program which makes liberal use of the CODE function. The code values that have remained constant are those in the range 12-17, 25-63, 140-145 and 153-191. Other assorted values have remained constant, but the main point is that the characters 0-9 and A-Z stay as they were.

8. A FOR/NEXT loop was always executed at least once. This may no longer be true, since if the "finish" value is already exceeded, the loop is now totally by-passed. Most likely this will not present any problems, but keep an eye open for loops which use variable names as the control and limit values.

Apart from that, you shouldn't have too many problems! The best option, really, is to understand the program (if it's not yours) and then re-write it properly. It'll work that much better.

ZX81 MODULE SELECTOR

This appendix lists the assembler module used in Section 7 "Using Machine Code". The addresses are all relative to the module base page address.

```
                         ;   ZX81 Module selector
                         ;
                         ;   System variables
                         ;   ----------------
                         ;
4004                    ramtop  equ   16388   ;top of memory pointer
4007                    ppc     equ   16391   ;current line number
400C                    dfile   equ   16396   ;pointer to display file
4010                    vars    equ   16400   ;pointer to variables
401C                    stkend  equ   16412   ;pointer to spare memory
407B                    spare   equ   16507   ;unused system variable
                        ;
0005                    maxfunc equ   5       ;number of routine functions
                        ;
0000'                           cseg
                        ;
0000'                   entry$point:
                        ;     On entry to a USR subroutine, registers BC
                        ;     contain the address of the routine. This
                        ;     is used throughout as the basis for all
                        ;     jumps & calls, effectively confining the
                        ;     routine to one page. Your RAMTOP must be
                        ;     set to a page boundary (i.e. a multiple of
                        ;     256). Register H is set to this page
                        ;     number, and should not be altered.
                        ;
                        ;     The system variable "SPARE" low-order byte
                        ;     is taken as the function call number.

0000'   3A 407B               ld      a,(spare)  ;load function number
0003'   FE 05                 cp      maxfuncs   ;is it a valid function?
0005'   D0                    ret     nc         ;return if invalid.
0006'   5F                    ld      e,a        ;create 16-bit offset
0007'   16 00                 ld      d,0
0009'   21 0010'              ld      hl,func$table   ;address of vectors
000C'   60                    ld      h,b        ;set page number
000D'   19                    add     hl,de      ;add in routine offset
000E'   6E                    ld      l,(hl)     ;get address offset
000F'   E9                    jp      (hl)       ;jump to function

0010'                   func$table:
                        ;     An entry value of zero in "SPARE"
                        ;     causes a jump to the first entry in this
                        ;     table. Each single-byte entry is the
                        ;     offset value of the routine address from
                        ;     the start of the module.

0010'   15                    defb    (func0-entry$point) and 0ffh
0011'   22                    defb    (func1-entry$point) and 0ffh
0012'   27                    defb    (func2-entry$point) and 0ffh
0013'   2C                    defb    (func3-entry$point) and 0ffh
0014'   31                    defb    (func4-entry$point) and 0ffh

0015'                   func0:
                        ;     Give estimate of available memory
0015'   21 0000               ld      hl,0       ;clear HL
0018'   39                    add     hl,sp      ;obtain stack pointer
0019'   ED 5B 401C            ld      de,(stkend);end of system memory
001D'   ED 52                 sbc     hl,de      ;get the difference
001F'   44                    ld      b,h        ;put answer into BC
0020'   4D                    ld      c,l
0021'   C9                    ret                ;return to BASIC
```

```
0022'                  func1:
                       ;      Give address of display file
0022'   ED 4B 400C            ld       bc,(dfile) ;load system variable
0026'   C9                    ret                 ;back to BASIC

0027'                  func2:
                       ;      Give address of BASIC variables
0027'   ED 4B 4010            ld       bc,(vars)  ;load contents of VARS
002B'   C9                    ret

002C'                  func3:
                       ;      Give address of some spare memory
                       ;   (beyond this routine)
002C'   01 0036               ld       bc,program$limit-entry$point
002F'   44                    ld       b,h
0030'   C9                    ret

0031'                  func4:
                       ;      Give current line number
0031'   ED 4B 4007            ld       bc,(ppc)   ;load from PPC sys. var.
0035'   C9                    ret

0036'                  program$limit equu     $  ;limit of memory

                              end
```

One final answer - the question was posed in the text with the "Silly Quiz" game. How can you answer each question correctly without even seeing the question?

Whenever a question is posed, the answer has already been set up in variable W$. By rubbing out the quotes and entering W$ as the solution, you will effectively enter the correct answer! Who needs questions anyway?

BASIC COMMAND SUMMARY

Name	Meaning
ABS(n)	Give modulus value of n
ACS(n)	Arccos value of n in radians
AND	Logical operator
ASN(n)	Arcsin value of n in radians
AT y,x	Forces print position to line y, column x
ATN(n)	Arctangent of n in radians
CHR$(n)	Gives character corresponding to code value n
CLEAR	Deletes all variables
CLS	Clears screen and sets print position to top left
CODE(s)	Gives code value of first character in string s
CONT	Resumes program execution after a report code
COPY	Copies screen content on ZX printer
COS(n)	Gives cosine value of n in radians
DIM	Dimensions an array (numeric or string)
EXP(n)	Supplies value of e^n (natural anti-log)
FAST	Places ZX81 into fast (or ZX80) mode
FOR	Introduces control variable of a program loop
GOSUB n	Obey subroutine at line n
GOTO n	Transfer program execution to line n
IF e THEN	Obeys THEN clause if expression e is true
INKEY$	Supplies character waiting at keyboard (if any)
INPUT	Inputs numeric or string expression from keyboard
INT(n)	Supplies next lower integer value of n
LEN(s)	Gives length of following string expression
LET	Assigns variables
LIST	Displays program text on screen
LLIST	Displays program text on ZX printer
LN(n)	Supplies value of $\log_e(n)$ (natural logarithm)
LOAD	Loads named program from cassette

LPRINT	Prints following expressions on ZX printer
NEW	Clears ZX81 completely
NEXT	Updates control variable to its next step value
NOT(n)	Inverts truth value of expression n
OR	Logical operator
PAUSE	Suspends execution for n/50 secs
PEEK(n)	Gives value of byte at address n
PI	Gives value of 3.1415926
PLOT	Blacks in pixel at screen coordinate x,y
PRINT	Places following expressions into display file
RAND	Sets random number seed
REM	Allows REMarks in program source
RETURN	Terminates a subroutine
RND	Supplies value of a random number 0<=n<1
RUN	Clears variables and commences program execution
SAVE	Saves a program on tape under a name
SCROLL	Shifts display up one line
SGN(n)	Supplies "sign" value of n (-1,0 or +1)
SIN(n)	Gives sine value of n in radians
SLOW	Places ZX81 in "compute-and-display" mode
SQR(n)	Gives square root value of n
STEP n	Alters step value of FOR loop to n
STOP	Suspends program execution, giving error report 9
STR$(n)	Gives string equivalent of numeric expression n
TAB(n)	Moves print position to column n
TAN(n)	Gives tangent value of n in radians
UNPLOT	Whitens the pixel at screen coordinate x,y
USR(n)	Calls machin code routine at memory address n
VAL(s)	Gives numeric equivalent of string s (if possible)

Useful information:

CODE "Ø" - 28
CODE "A" - 38

FRAMES	16436/7	VARS	16400/1
RAMTOP	16388/9	DFILE	16396/7

Program starts at 16509

$9.95

XI/83

ERROR CODES

Error code	Meaning
0	Successful completion of program.
1	Control variable does not exist.
2	Undefined variable name.
3	Subscript out of range.
4	Not enough memory.
5	Screen display file full.
6	Arithmetic overflow.
7	RETURN found with no GOSUB.
8	INPUT used in direct mode.
9	STOP command executed.
A	Invalid function argument (e.g. SQR -1)
B	Integer out of range (PRINT AT,PLOT,etc.)
C	VAL argument is not a numeric expression.
D	Program interrupted (BREAK or STOP in INPUT)
E	Unused.
F	No name given to SAVE command.

THE ZX 81 POCKET BOOK

TREVOR TOMS

The ZX-81 computer from Sinclair Research, Ltd., is an exciting breakthrough in personal computing. About the size of this book, it uses your television set for display and any cassette recorder to save programs. Though it can be used for games, for home recordkeeping, and for business functions, it is not "for" any of these uses. Because it is the least expensive, complete, powerful computer on the market, it is an ideal "first computer," to introduce adults and children to the world of computing.

The ZX-81 Pocket Book contains programs ready to run, as well as programming hints to help you create your own programs. **The ZX-81 Pocket Book** also includes an introduction to machine code, has a complete adventure game entitled "City of Alzan" and guides for you to create your own adventure games.

Information about the **ZX-81** can be obtained from the National ZX Users Group, 599 Adamsdale Rd., N. Attleboro, Mass. 02760

cover design by: Joyce Thompson

RESTON PUBLISHING COMPANY, INC.
A Prentice-Hall Company
Reston, Virginia

0-8359-9524-0