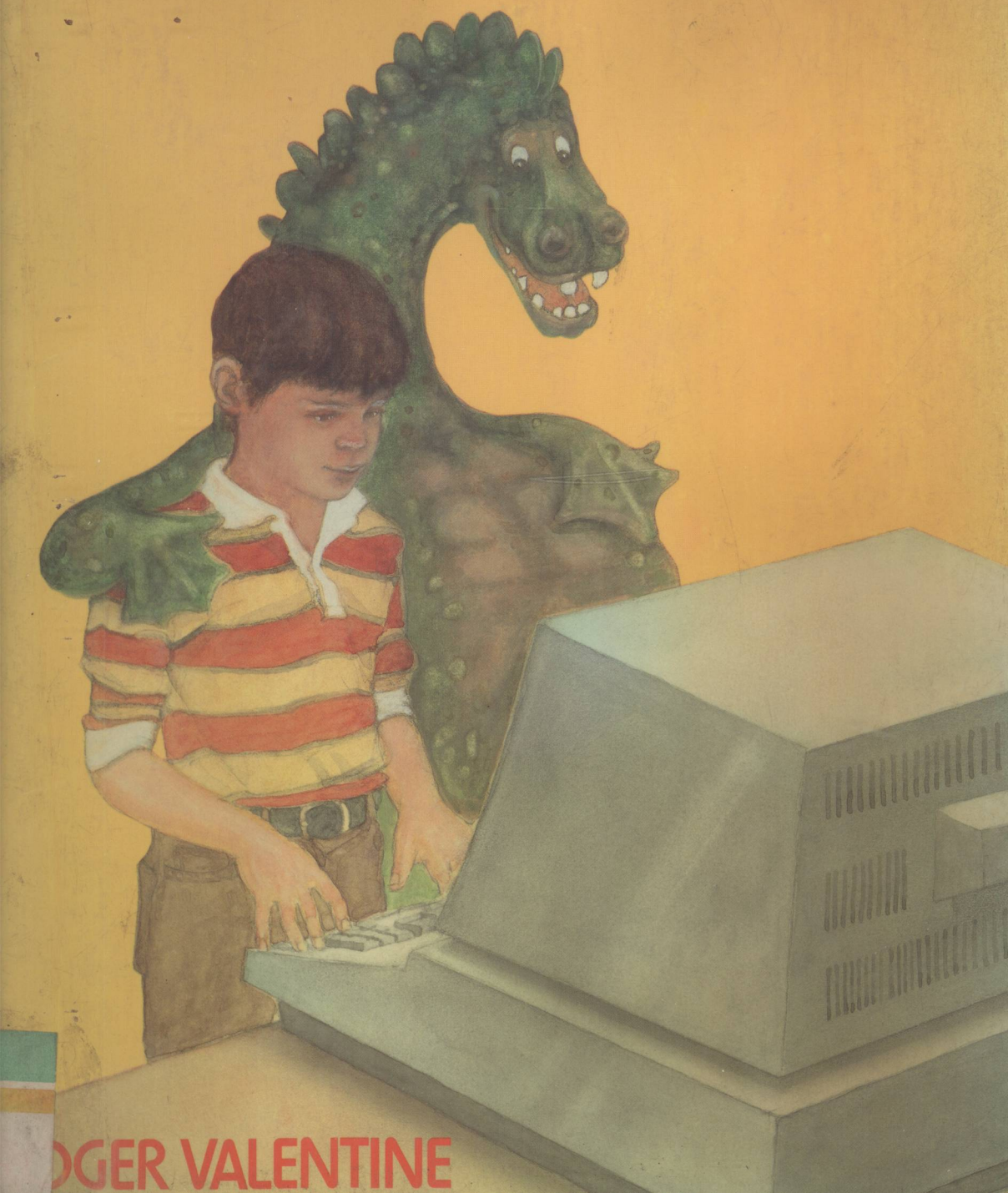
COCO™ EXTRAVAGANZA



OGER VALENTINE

CoCo Extravaganza

Roger Valentine

A Wiley Press Book

John Wiley & Sons, Inc.

New York • Chichester • Brisbane • Toronto • Singapore

Publisher: Judy V. Wilson
Editor: Theron Shreve
Managing Editor: Katherine Schowalter
Composition and Makeup: The Publisher's Network

Library of Congress Cataloging in Publication Data

Valentine, Roger, 1949-
CoCo extravaganza.

(Recreational computing series)
Includes index.
1. Dragon 32 (Computer)—Programming. 2. Basic (Computer program language) I. Title II. Series.
QA76.8.D73V35 1984 001.64'25 83-26009
ISBN 0-471-80034-1

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

INTRODUCTION

CoCo Extravaganza is a book of 50 ready-to-run programs for your 64K Color Computer™ with at least one disk drive, but it is much more than that. It is also a complete course in Color Computer Extended BASIC programming.

By keying in the programs AND reading the accompanying documentation, you will learn how and why the programs work and how you can write such programs yourself.

The documentation for each program is in two parts. The first part explains what the program does and how you should use it. The second part, more importantly, is headed *Program Notes* and explains how the program is constructed and what the lines of BASIC mean.

New programming concepts will be introduced gradually as you progress through the book. In *Numerology*, for instance, it is assumed that you already know the meaning of PRINT and INPUT, etc., but more complex functions such as ASC and MID$ are explained. Before you have completed the first section, these expressions will seem commonplace to you, as will SOUND, TIMER, INKEY$, and many others. The *Starters* section also includes an introduction to medium- and high-resolution graphics and other aspects of programming, which will be covered more fully later on.

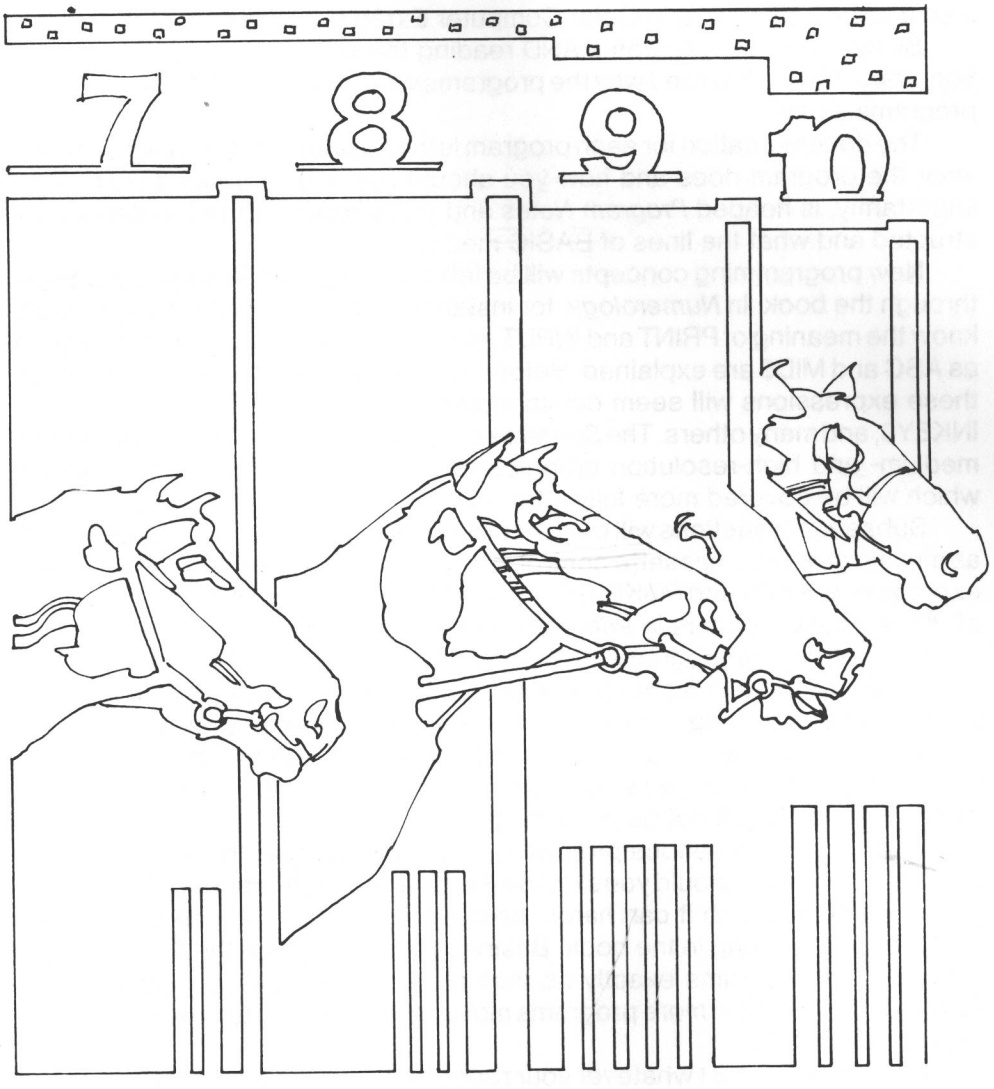
Subsequent sections will be concerned with more difficult concepts, such as moving graphics, cassette control, and those most frightening of all BASIC expressions, PEEK and POKE. (I don't want to put you off, but the first mention of POKE actually occurs in *Primes*, the fifth program in the *Starters* section.)

But what of the programs themselves? What are they like?

Naturally, there is a strong emphasis on games - even the "educational" programs have a strong games-playing element. (My own preference is for the more sedentary type of "thought-provoking" game, and these are well represented, but those of you who prefer annihilating aliens and exterminating innocent civilians will not be neglected!)

There are more serious programs, too, particularly in the *Business* section. On no account should you skip the *Bar Charts, Pie Charts, and Graphs* program, which, although it can hardly be described as fun, is one of the most useful and interesting in the book. Business users may use the *Invoicing* and *Sales Ledger* programs exactly as they are printed and may, hopefully, be inspired to write some more programs along the same lines to create their own accountancy suite.

All in all, I think that whatever your reasons for using a personal computer, you will gain a great deal from this book. But that is enough reading for now; it is time to get to the Color Computer keyboard and start entering some programs!



CONTENTS

CHAPTER

1

YOUR STARTERS FOR TEN

- NUMEROLOGY
- STOPWATCH
- ACT YOUR AGE
- PIANO
- PRIMES
- I CHING
- KALEIDOSCOPE
- PAINT BOX
- SKETCH PAD
- PRINT @ WHERE?

NUMEROLOGY

It is claimed by some people that by converting your name into a single-digit number (using the rule A=1, B=2, Z=26=2+6=8, etc.), the number obtained is an indication of your type of personality. You can also use this method to determine your own particular "lucky number" or to assess your compatibility with others. (If their number is too far removed from yours, forget it!)

Program Notes

The program uses some of the Color Computer's string handling facilities to extract each character in turn from the name that has been entered (N$).

The loop (lines 50–100) works as follows:

Line 50 sets the limit of the loop to the number of characters in N$. (Line 30 has already ensured that N$ is not a "null string," i.e., does not contain zero characters.)

The expression MID$ (N$, J, 1) in line 60 extracts the Jth character from N$, and ASC produces the ASCII code, which the Color Computer uses to represent this character. The Color Computer manual, GOING AHEAD WITH EXTENDED COLOR BASIC, gives a list of these codes, and you will notice that uppercase letter A has Code 65, so that by subtracting 64 from the ASC value, we end up with C= a number representing the position in the alphabet of the character.

Notice that line 60 performs these three steps - extract a character, convert to ASC value, subtract 64 - in a single expression. This could have been written:

```
60 T$=MID$(N$,J,1)
61 C=ASC(T$)
62 C=C-64
```

Sometimes, in the interest of clarity, it is preferable to break down a complex expression in this way but, in general, such inefficient programming should be avoided.

Line 70 ensures that the character is, indeed, a letter of the alphabet so that spaces, hyphens, apostrophes, etc., do not affect the result, and line 80 reduces the number "modulo 9" so that a single digit is obtained. (Line 80 is unusual in that it repeatedly calls itself until such time as a single digit is produced.) Similarly, line 90 does the same thing to the "running total," N.

NUMEROLOGY

The program ends rather unusually by looping back to the beginning (missing only the *clear screen* instruction in line 10), ready for the next name. RUN is a perfectly valid expression to use within a program.

(Note, however, that if you RENUMber this program, the number 20, which follows the word RUN, is not adjusted to match the new number of the second program line.)

```
10 CLS
20 PRINT@ 384,"ENTER YOUR NAME"
30 INPUT N$: IF N$="" THEN 20
40 C=0: N=0
50 FOR J=1 TO LEN(N$)
60 C=ASC(MID$(N$,J,1))-64
70 IF C<1 OR C>26 THEN 100
80 IF C>9 THEN C=C-9: GOTO 80
90 N=N+C: IF N>9 THEN N=N-9
100 NEXT J
110 CLS: PRINT"NAME:-": PRINT@ 65,N$: PRINT@ 160,"NUMBER:-":
    PRINT@ 224,N
120 PRINT@320,"NEXT ONE PLEASE": RUN 20
```

STOPWATCH

The Color Computer's TIMER function gives access to an internal clock that is accurate to 1/60 of a second. Some of the more obvious uses of TIMER are to impose time limits in games or quiz programs, to create a digital clock display, or even to simulate a conventional-style clock using high-resolution graphics.

This program illustrates a simple use of TIMER in conjunction with a "reaction test."

After a random pause, an alpha-character (A-Z) appears on the center of the screen, and you have to press the corresponding key as quickly as possible. If you constantly take more than one second to hit the key, then either you have incredibly slow reactions or you need to become more familiar with the Color Computer's keyboard. Try the *Keyboard Test* program on page 44.

Program Notes

TIMER works by using two memory locations (274 and 275) in the area of the Color Computer's RAM reserved for "System Use." The contents of location 275 are incremented every 1/60 second until they attain a value of 255, after which they are reset to zero. (If you are familiar with 8-bit binary arithmetic, you will appreciate that 255 plus one DOES equal zero, as the "carry" is ignored.) Every time this happens, the contents of 274 are incremented by one. Thus, the contents of 274 change every 256/60 (approximately 4.27) seconds. The expression TIMER=0 sets the contents of BOTH locations to zero, and the value of TIMER will continue to increase for 4.27*256 seconds (just over 18 minutes), when both locations will once again contain zero.

Line 30 of the program selects a random number in the range 1 – 26 and adds 64 to produce the ASCII code of an alphabetic character. CHR$, in line 40, is the reverse of ASC, as used in the previous program, and converts the code value into its corresponding character. SOUND 255,1 produces a very short, high-pitched beep fractionally before the character is PRINTed and the TIMER is started.

Line 60 merely waits until a key is pressed before reading the current value of TIMER. The value of INKEY$ is always "" (the null string) until a key is pressed, at which time INKEY$ returns the character represented by the keypress. Remember that INKEY$ only retains this character momentarily and then returns to "" again. Therefore, if, as in this case, you want to use the character later, you must set a string variable (conventionally I$ is used for this purpose, but any variable will do) equal to INKEY$ as soon as it attains any value other than "".

STOPWATCH

INKEY$ is used again in line 100. Here, there is no need to use I$ because as soon as INKEY$ equals the SPACE character (CHR$(32)), the program is restarted.

IMPORTANT NOTE: The actual rate at which the TIMER is incremented depends on the frequency of the main power supply. It is assumed that the power being used is 60 cycles per second (60 hertz). The TIMER is incremented every 1/N seconds, where N is the frequency of your power supply in hertz.

```
10 CLS: PRINT"PRESS KEY:-"
20 FOR J=1 TO RND(1000)+100: NEXT J
30 X=RND(26)+64
40 SOUND 255,1: PRINT@ 271,CHR$(X)
50 TIMER=0
60 I$=INKEY$: IF I$="" THEN 60 ELSE T=TIMER
70 IF I$<>CHR$(X) THEN PRINT@ 362,"WRONG KEY!": SOUND 5,5:
   GOTO 90
80 PRINT@ 352,"YOUR TIME:";T/60;"SECONDS": SOUND 200,5
90 PRINT@ 451,"PRESS SPACE TO TRY AGAIN"
100 IF INKEY$=CHR$(32) THEN RUN ELSE 100
```

ACT YOUR AGE

This is an old party trick that never fails to confound people. You are presented with six tables of numbers and must indicate which of the tables includes your age. From this sparse information, the Color Computer will instantly and with unvarying accuracy deduce how old you are. Due to the lack of space on the Color Computer's screen, the program will only work for ages in the range of 0 – 60. (My apologies to the over-60s.) It would, in any case, be impossible to calculate ages over 63 using only six tables.

How It Works

Computer buffs should be able to see the secret of this trick quite easily. If you convert your age to a six-digit binary number, e.g., 15 becomes 001111, and then imagine the six tables being numbered 1 – 6, each table represents one digit of the binary number. If the digit is a 1, then the (decimal) number is included in that list, and if it is a 0, then the number is absent. Thus, 15 would be absent from tables one and two and present in the other four tables. When you enter your yes/no responses, what you are in effect doing is entering your age as a binary number (N indicating a 0, Y indicating a 1), and the computer merely has to convert this back into decimal.

Program Notes

This program introduces two essential concepts in programming: arrays and subroutines. Subroutines may be used to avoid repetitiveness whenever a particular section of program is required several times or, as here, merely to isolate a particular procedure that is not central to the program itself. Arrays can be one-dimensional (as is P in line 20), representing a list of numbers, or two-dimensional (B in line 20), representing a table. For more complex programs, they may have even more dimensions.

Array B represents the six "tables" of numbers. (They are not really tables; they just look like tables because of the way they are displayed on the screen. The entire set of numbers is a single table comprised of six lists.) Array P is used as a pointer to indicate the next vacant element in each of B's lists.

The initializing process (lines 10 – 80) involves converting each of the numbers 0 – 60 into binary and then updating the appropriate tables according to the rule explained above. Computers are very much at home dealing with binary numbers but, unfortunately, the BASIC language is designed for the benefit of decimal-orientated humans, so the only way that binary numbers can be represented is to treat them as strings of characters. The process of converting numbers to binary form has, therefore, been delegated to the subroutine starting at line 1000.

ACT YOUR AGE

Line 1010 extracts each binary digit of the number (by dividing by successive powers of 2) and places the resulting digit into the string A$. So X is the particular binary digit, 0 or 1; STR$(X) is the same digit, now treated as a string rather than a number. RIGHT$(STR$(X),1) is the right-most character of the string STR$(X) (which is necessary only because of the way STR$ works on the Color Computer: STR$(1) is not "1", as you might expect, but " 1" with a leading space inserted in front of the required character). The expression A$=A$+RIGHT$(STR$(X),1) does not ADD the new digit to the old one, it CONCATENATES the new string onto the end of the old one. So, at the end of the subroutine, A$ is a six-character string containing 0s and 1s, i.e., the binary representation of the original number. Line 70 then extracts each character of A$ in turn and, if it is a 1, inserts the original decimal number into the appropriate list, as well as updating the appropriate element of the pointer array.

```
10 CLS: PRINT"INITIALIZING: PLEASE WAIT"
20 DIM B(6,29),P(6)
30 FOR J=1 TO 60
40 Y=J
50 GOSUB 1000
60 FOR K=0 TO 5
70 IF MID$(A$,K+1,1)="1" THEN B(K,P(K))=J: P(K)=P(K)+1
80 NEXT K,J
90 AGE=0
100 FOR K=0 TO 5
110 CLS: PRINT"IS YOUR AGE IN THIS LIST? (Y/N)"
120 FOR J=0 TO 29: IF B(K,J)<>0 THEN PRINT@ 16*J+32,B(K,J);
130 NEXT J
140 I$=INKEY$
150 IF I$<>"Y" AND I$<>"N" THEN 140
160 IF I$="Y" THEN AGE=AGE+B(K,0)
170 NEXT K
180 CLS: PRINT"YOUR AGE IS";AGE
190 PRINT@ 288,"ANOTHER GO?"
200 IF INKEY$="Y" THEN 90
210 IF INKEY$="N" THEN END ELSE 200
1000 A$="": FOR K=5 TO 0 STEP-1
1010 X=INT(Y/INT(2^K)): A$=A$+RIGHT$(STR$(X),1)
1020 Y=Y-(X*INT(2^K))
1030 NEXT K
1040 RETURN
```

PIANO

Turn your Color Computer keyboard (the top two rows of it, at least) into a two-octave piano.

The row of alpha keys (including the three ARROW keys and the @) become the "white notes," and the numeric keys (including the colon and minus sign but NOT including the BREAK key) become the "black notes." The lowest key (the up arrow) has been defined as the note D, an octave above middle-C, but you can easily redefine the keyboard if you like. Changing the octave is particularly easy— just change the 3 in line 10 to 1 or 2. (If you change it to 4 or 5, you will not be able to use all the keys as the Color Computer will only play notes in a five-octave range.)

Remember that as there are no "black notes" on a piano between B and C or between E and F, some of the numeric keys are undefined (i.e., play no notes). These are keys 2, 6, 9, and BREAK. (You see now why the first note on the keyboard is D. The BREAK key is not required as a note and so can retain its normal function of stopping the program.)

Program Notes

The string in line 20 contains all the keys that are required for the piano keyboard. The left and right arrow keys cannot be typed directly into the program as they are normally cursor control keys, and they do not produce arrows when typed. However, it is possible to enter them into the string by referring to their ASCII values (8 and 9, respectively).

The string array P$ is used to contain the 23 strings of notes to be PLAYed. As each key plays a single note, the PLAY strings consist of nothing more than an "O" followed by a number (to set the octave), a semicolon as a separator, and a number in the range 1–12 defining the note. You can see how each of the PLAY strings is built up in line 80.

The array N is very largely unused. Whenever an array is DIMensioned, all its elements are set to zero (or to the null string, in the case of a string array), and although it is usual to assign other values to the elements later, it is by no means essential to do so. (Very often it is convenient to leave the "zeroth" element of an array unassigned as we humans usually prefer to start counting at 1 rather than zero.) So array N is DIMensioned to be large enough to hold every possible ASCII code but, in fact, only the elements that correspond to a code belonging to one of the characters in N$ are given the value of that code; all the others remain at zero.

PIANO

Lines 50 and 130 both rely on conditional expressions. A conditional expression is one that may be either TRUE or FALSE. In line 50, for instance, the expression J>11 will be FALSE for the first eleven journeys through the loop (FOR J=1 TO 23) and TRUE thereafter. J=23 will always be FALSE until the very last journey through the loop. The computer assigns numeric values to expressions that are TRUE or FALSE. A TRUE expression has value -1, and a FALSE expression has value 0. So line 50 is a rather neater way of saying:

IF J>11 THEN O=OCT+1: IF J=23 THEN O=OCT+1

In line 130, the actual condition appears to be missing, but the computer will regard ANY expression with a value of 0 as FALSE and any other value (negative or positive) as TRUE. Hence, when you press a key for which the corresponding element of array N has been left unassigned (i.e., which retains its original value of zero), the expression N(I) will be FALSE and no note is PLAYed.

```
10 OCT=3
20 N$="^1QW3E4R5TY7U8I0OP:;-"+CHR$(8)+CHR$(9)
30 DIM P$(23),N(255)
40 FOR J=1 TO 23
50 O=OCT-(J>=11)-(J=23)
60 T=J+2
70 IF T>12 THEN T=T-12: GOTO 70
80 P$(J)="O"+STR$(O)+";"+STR$(T)
90 N(ASC(MID$(N$,J,1)))=J
100 NEXT J: CLS 3
110 I$=INKEY$: IF I$="" THEN 110
120 I=ASC(I$)
130 IF N(I) THEN PLAY P$(N(I))
140 GOTO 110
```