Advanced Programming for the ORIC

Gerard Mason

Advanced Programming for the Oric

Gerard Mason

McGRAW-HILL Book Company (UK) Limited

London · New York · St Louis · San Francisco · Auckland · Bogotá
Guatemala · Hamburg · Johannesburg · Lisbon · Madrid · Mexico
Montreal · New Delhi · Panama · Paris · San Juan · São Paulo
Singapore · Sydney · Tokyo · Toronto

Published by
McGRAW-HILL Book Company (UK) Limited
MAIDENHEAD · BERKSHIRE · ENGLAND

British Library Cataloguing in Publication Data

Mason, Gerard
Advanced programming for the Oric
1. Oric 1 (Computer)
I. Title
001.64'04 QA76.8.07
ISBN 0-07-084745-2

Library of Congress Cataloging in Publication Data

Mason, Gerard.
Advanced programming for the Oric.
Bibliography: p.
Includes index.
1. Oric 1 (Computer) — Programming. 2. Basic (Computer program language) I. Title.
QA76.8.068M37 1984 001.64'2 84-12557
ISBN 0-07-084745-2

1 2 3 4 5 CUP 8654

Printed in Great Britain at the University Press, Cambridge

PREFACE

In some books for Oric an attempt is made to rewrite the manual. Other books tell you how to make little green aliens appear, and then how to zap them. Few if any show you how you can use your computer in ways that really stretch it; how to make it do useful work; how to learn about computers and about music, art and mathematics while still having fun. Until now.

In this book the author aims to give you programs which are useful, interesting, educational, *and* entertaining in their own right, but which will also show you how to tackle a project of medium complexity with every chance of success by firstly defining the problem and then breaking it up into smaller parts which, on their own, are easy to solve. He aims also to make you familiar with the key concepts of several fields so that you will be able to formulate interesting projects to solve, particularly with regard to music and graphics.

As some of the programs are quite long there is a cassette available which contains most of the programs in the book. Also, those programs which run slowly in BASIC have had sections rewritten in machine code for speed. All programs can be listed, and the machine code sections are accompanied by a disassembly as well as comments in order to make them easy to understand.

CONTENTS

1 CHARACTERS AND TEXT

1.1 Strings

A *string* is a sequence of characters, which may be letters, digits, punctuation marks, graphics symbols — even commands to perform actions such as turning the cursor off. The teletext attributes can also be held as strings.

Oric will accept anything you can put between quotation marks as a valid string.

A variable of the form A$, B7$, or AZ$ can be used to stand for a string. Anywhere you could have typed the string you can use the variable. Remember, though, that only the first two characters in the variable's name are used to identify it so Oric will treat NAME$ and NAVY$ as the same:

```
NAME$="Nelson":NAVY$="British  :PRINT
NAME$:PRINT NAVY$
British
British
```

1.1.1. Formatting

Notice how in the example above the strings were printed on separate lines. You can stop this by putting a semicolon (;) after the string or string variable in the PRINT statement. A comma does the same with the added effect that it will introduce four spaces between the items:

```
PRINT NAME$;NAVY$
BritishBritish
PRINT NAME$,NAVYs
British      British
```

1.1.2. Plot

Using the PLOT command you can specify exactly where a character or a string will appear.

The basic form of the command is PLOT, X,Y,Z, where X is the distance across the screen (Ø to 38, left to right) and Y is the

distance down (Ø to 26). Z is the code for the character to appear, and can be between Ø and 255.

X, Y, and Z can be calculated as well as specified explicitly. For instance,

```
FOR N=3 TO 9:PLOT N*3,N+2,N+65:NEXT
```

There are other ways of saying exactly what is to be plotted. Instead of Z you can say CHR$(Z), and even complete strings. So if M$="Honest Oric" then both

PLOT 9,9,M$ and PLOT 9,9,"Honest Oric"

will put the string onto the screen, starting at 9,9 and ending at 19,9.

1.1.3 String Operators

To help in managing strings there are a number of words which deserve a brief description.

LEN() with either a string in quotes or a string variable between the brackets will return the *length* of that string. So if A$="STRINGY" then the command A=LEN(A$) will make A equal 7. PRINT LEN("BACON") will result in a 5 being printed.

VAL() returns the value of a string, so VAL("123") is 123. VAL("XYZ") is zero.

STR$() works in the other direction. So if X=456 then A$=STR$(X) will have the same effect as A$="456". Notice that there is a space at the beginning (on the Atmos; Oric 1 has CHR$(2) instead) so that A$ is actually *four* characters long. This has the unfortunate effect that VAL(A$) here is zero! There are ways to get around this.

CHR$() turns code numbers into characters, according to the ASCII code. It works with numbers from Ø to 255 inclusive, though those below 32 have special functions and those above 127 are mostly repeats of the others.

Three more important words are LEFT$, MID$, and RIGHT$ which slice strings into parts in the way their names suggest.

If A$="ABCDEF" then we can use LEFT$ to extract a certain number of the left-most characters. For instance, B$=LEFT$(A$,3) will set B$="ABC". The number can be Ø — in which case B$ would be empty — or it can be greater than the length of A$ — in which case B$=A$.

Right-hand slices can be made in the same way. Typing C$=RIGHT$(A$,3) makes C$="DEF".

The most useful of the three is MID$. It gives a string starting at a certain position, of a certain length. So D$=MID$(A$,2,3)

makes D$="BCD": *three* characters of A$ starting at position *two*.

We can use MID$ to get rid of the difficulty with STR$. If A=789 then we set T$=STR$(A), so that T$="789". Then simply make A$=MID$(T$,2), after which A$="789". Notice that leaving out the length number from MID$ makes it assume that we want the whole thing. Now VAL(A$) is 789.

You can stick two or more strings together using +. If A$="abc" and B$="xyz" then C$=A$+B$ will make C$="abcxyz". Similarly, D$=A$+" "+B$ makes D$="abc xyz".

< and > compare strings alphabetically. < means 'is before' and > means 'is after'. So "AAA" < "AAB" and "CZ" < "D".

1.2. **The ASCII code**

This is the American Standard Code for Information Interchange, and associates characters with numbers. It is used by CHR$() and < and >. There is a copy of the table in Appendix C on page 153, but if ever you need to know what the ASCII code of a character is, the word ASC() will do it for you. ASC("A") is 65, ASC("B") is 66, and so on.

As you can see if you look at the table, some numbers are associated with actions rather than shapes. So PRINT CHR$(12) will clear the screen. Characters below 32 can be included in strings, so if A$=CHR$(12)+"Hi there!" then PRINT A$ will clear the screen before printing the message. Try A$="Mary had little lamb"+CHR$(10)+CHR$(13)+"its fleece was white as snow". PRINT A$.

1.2.1 Ciphers

It is easy to get Oric to play about with the ASCII values of a string in order to produce an enciphered version. For example, if A$="The enemy has retreated" we can produce an enciphered version with:

```
B$="":A$="The enemy has retreated"
FOR N=1 TO LEN(A$):B$=B$+CHR$(ASC(MID$(A$,N)
)+1):NEXT
```

All this does is add 1 to the value of each letter, so that T becomes U, h is now i, e is f, and so on. Print out B$.

This little program produces a very secure encipherment of an input string. It is based on the fact that Oric's random-number generator is in fact anything but random! It starts the generator off in line 2Ø with a negative number. Each number produced

after that can be duplicated by starting it off once again with the same negative number. You simply choose a number to encipher a message and use the same one to decipher it. This forms the 'key'. The same message will be enciphered differently if a different key number is used!

```
10 INPUT "Code Number ";N
15 INPUT "En/Decipher ";C$
20 R=RND(-N):C=1
25 IF C$="D" THEN C=-1
30 INPUT T$:PRINT "  ";
35 FOR F=1 TO LEN(T$)
40 T=ASC(MID$(T$,F))-ASC("A")
45 IF T<0 THEN 60  'only encipher letters
50 T=T+INT(RND(1)*26)*C
55 T=T-INT(T/26)*26
60 PRINT CHR$(T+ASC("A"));
65 NEXT:PRINT:PRINT:GOTO 30
```

The code number can be any positive number, including decimals; thus 1.2345 is just as good as, but different from, 12345. One further interesting feature is that *superencipherment* is possible. Just take the first encipherment and feed it in again. Obviously, the text must also be *deciphered* twice!

1.3 *Number Cruncher*

Here is a program to show some of the truly amazing uses to which strings can be put. It is the first major program in the book.

One of the principle failings of small computers is that they cannot deal accurately with very large or very small numbers. The program has been designed to deal with this problem. You can add, subtract, multiply, divide, and raise to powers whole numbers of up to 255 digits, without loss of accuracy. I have given it a flavour of the FORTH programming language in the method of entering numbers and commands. If you understand how to use it, and it is not difficult, you will have made a good start towards mastering FORTH.

On running the program the "?" prompt appears. This means it is ready for a number or a command. Suppose you want to find 23 times 24. Type 23 24 * . and press RETURN. Oric will immediately answer +552 followed by "OK", and the prompt reappears. When entering numbers you must leave at least one space after each

one. You can insert as many spaces as you like, or even enter each number on a new line.

Obviously, the answer you got was the same as if you had typed PRINT 23*24 using ordinary BASIC. So why does the asterisk come *after* the second number and what does the full stop mean?

The answer is contained in the fact that the program, like FORTH, uses a *stack* for its arithmetic. A stack is just another way to store numbers, with the property that the last number onto the stack is the first one off. The spring-loaded devices used in cafes to store plates are an example of a stack in action. See Fig. 1.1 for an illustration of what happens when two numbers are *pushed* onto a stack and then *popped* off it.

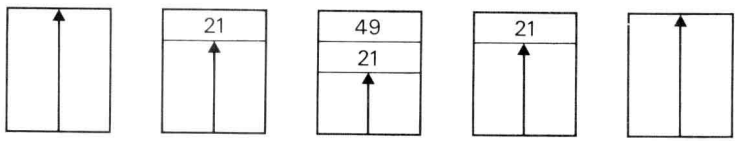

Figure 1.1

However, when we ran the program we multipled the numbers and printed the answer. Figure 1.2 shows what happened: the "*" popped the top two numbers, multiplied them, and pushed the result back onto the stack. The "." popped the top number and printed it.

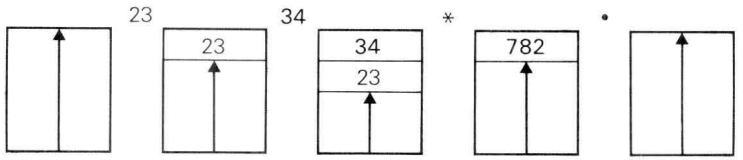

Figure 1.2

Note that it is not necessary for the stack to have *only* two numbers on it for "*" or one number for "."; it is merely necessary for it to have *at least* that many. Notice also that printing a number removes it from the stack.

Since the program is fairly long it has been written in modules so that you only need to type one in when you want it. For this reason, type them in the order given, since later ones use earlier ones as subroutines.

The first module is concerned with setting up the program, and entering numbers and commands.

```
100 CLS:GRAB:HIMEM #B400
105 Z$="0":Z=ASC(Z$):T=10
110 S$=" ":E$="":MAX=120
115 P=0:DIM ST$(MAX)
120 REPEAT:INPUT I$
125 T$=LEFT$(I$,1):IF T$=E$ THEN 295
130 IF T$=S$ THEN 315
135 IF T$<Z$ OR T$>"9" THEN 160
140 N=0:REPEAT:N=N+1
145 UNTIL MID$(I$,N,1)=S$ OR N>LEN(I$)
150 A$=LEFT$(I$,N-1):I$=MID$(I$,N)
155 GOSUB 410:GOTO 315
160 IF P=0 THEN 290
165 Y$=ST$(P-1)
170 IF T$="=" THEN 470
175 IF T$="-" THEN 490
180 IF T$="." THEN 515
185 IF P=1 THEN 290
190 P=P-2:X$=ST$(P):ST$(P)=E$
195 ST$(P+1)=E$:F=FRE("")
200 IF T$="\" THEN 535
205 X=44-ASC(X$):Y=44-ASC(Y$)
210 X$=MID$(X$,2):F=FRE("")
215 Y$=MID$(Y$,2):F=FRE("")
220 IF T$<>"+" THEN 235
225 GOSUB 560:X$=E$:Y$=E$:X=44-X
230 F=FRE(""):GOSUB 445:GOTO 315
235 IF T$<>"*" THEN 250
240 GOSUB 650:GOSUB 445:GOTO 315
250 IF T$="/" OR T$="%" OR T$="¦" THEN 755
280 IF T$="↑" THEN 900
285 PRINT "Input Error":GOTO 295
290 PRINT "Stack Underflow"
295 PRINT "OK":F=FRE(""):UNTIL FALSE
300 :
```

1Ø0: Owners of 16K machines should set HIMEM to #34Ø0 and set MAX=9 in line 11Ø.

The module scans input strings to see if they are numbers or sequences of commands. Line 16Ø checks to see that the stack has at least one operand for the operations of "=" (duplicate number on top-of-stack), "−" (negate top-of-stack), and "." (print top-of-stack). From now on I shall call the number on the top of the stack 10S, the number below it 20S, the one below that 30S, etc.

Line 185 checks that there are at least two numbers on the stack for the operations of "\" (swap 10S and 20S around), "+" (add 10S and 20S), "*", "/" (divide 20S by 10S), "%" (find the remainder on dividing 20S by 10S), and "¦" (find the quotient *and* remainder), and "↑" (raise 20S to the power of 10S).

The next block consists of five subroutines split into two distinct groups: find the next item in the input string and add a number to the stack. The second of these is the longer and comprises the basic housekeeping routine of the entire program.

```
310 REM get next item in I$
315 I$=MID$(I$,2):GOTO 125
320 :
325 REM increment the stack pointer
330 IF P<MAX THEN P=P+1:RETURN
335 PRINT "Stack Overflow"
340 POP:POP:GOTO 285
345 :
350 REM remove leading zeros
355 A$=Z$+A$:REPEAT
360 F=FRE(""):A$=MID$(A$,2)
365 UNTIL LEN(A$)=1 OR ASC(A$)>Z
370 RETURN
375 :
380 REM check digits
385 REPEAT:F=ASC(MID$(A$,N)):N=N-1
390 UNTIL N=0 OR F<Z OR F>(Z+9)
395 RETURN
400 :
405 REM add a number to the stack
410 X=ASC("+")
415 N=LEN(A$):GOSUB 385
420 IF N=0 THEN 445
425 POP:PRINT:PING:PRINT A$
430 FOR J=0 TO N-1
435 PRINT S$;:NEXT
440 PRINT "↑":PRINT:GOTO 285
445 GOSUB 330:GOSUB 355
450 ST$(P-1)=CHR$(X)+A$
455 A$=E$:F=FRE(""):RETURN
460 :
```

Next come the stack operations proper. They are an intermediate level between the housekeeping routines and the

arithmetical ones. The options are to duplicte the current top-of-stack, negate it, print it, and swap it with the number immediatley below it (20S). The operations are invoked by "=", "–", ".", and " \" respectively (see Fig. 1.3).

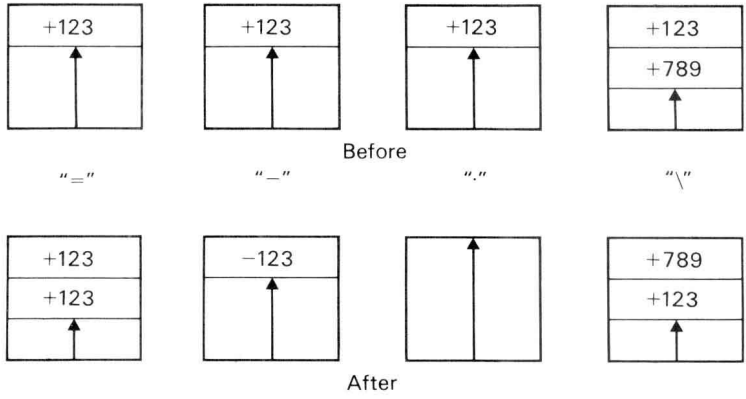

Figure 1.3

```
465 REM duplicate TOS
470 GOSUB 330:ST$(P-1)=Y$
475 Y$=E$:F=FRE(""):GOTO 315
480 :
485 REM negate TOS
490 T$=CHR$(88-ASC(Y$))
495 ST$(P-1)=T$+MID$(Y$,2)
500 F=FRE(""):GOTO 315
505 :
510 REM print TOS
515 P=P-1:PRINT Y$:ST$(P)=E$
520 Y$=E$:F=FRE(""):GOTO 315
525 :
530 REM swap TOS and 20S
535 ST$(P)=Y$:Y$=E$
540 ST$(P+1)=X$:X$=E$
545 F=FRE(""):P=P+2:GOTO 315
550 :
```

Here is the first of the arithmetical routines — addition. It can also be used for subtraction; simply negate the number you want to subtract and then add it. So 789–123 becomes 789+(–123). The routine is called by the multiplication and power routines.

```
555 REM addition subroutine
560 S=X*Y:C=0:A$=E$:F=FRE("")
565 L=LEN(X$)-LEN(Y$)
570 IF L>0 THEN 580
575 IF X$<Y$ OR L<0 THEN H$=X$:X$=Y$:Y$=H$:
    X=Y:L=-L:H$=E$:F=FRE("")
580 FOR N=LEN(Y$) TO 1 STEP -1
585 D=VAL(MID$(Y$,N,1))*S+VAL(MID$(X$,N+L,
    1))+C
590 C=(D<0 OR D>9)*-S:D=D-10*C
595 A$=CHR$(D+Z)+A$:F=FRE(""):NEXT
600 X$=LEFT$(X$,L):IF C=0 THEN 635
605 IF L=0 THEN 630
610 REPEAT:D=VAL(RIGHT$(X$,1))+C
615 C=(D<0 OR D>9)*-S:D=D-10*C:L=L-1
620 A$=CHR$(D+Z)+A$:X$=LEFT$(X$,L)
625 F=FRE(""):UNTIL X$=E$ OR C=0
630 IF C THEN A$="1"+A$
635 A$=X$+A$:RETURN
640 :
```

The routine always adds the smaller number to the larger. If they are both negative or both positive this makes no difference, but it is vital if they are of different signs.

560: S=1 if the numbers are both positive or both negative, so that corresponding digits are *added* in line 585. If they are of different signs then S=-1 and one is subtracted from the other. C is the carry.

570–575: These lines make sure that the smaller number is added to the larger.

590: Adjusts if a digit would be above 9 or below Ø and makes carry=1 or borrow=-1 as appropriate.

600ff: The first part stops when all the digits of the smaller number have been added, so that if the sum is 1234567890987654321+1 then only *one* addition is made. The second part takes over if there is still a carry at this stage, so that an addition such as 9999999999+1 is done correctly.

Now for the multiplication routine. It performs a long multiplication in a way similar to that used when doing it by hand.

```
645 REM multiplication subroutine
650 M=X*Y:S=1:L=LEN(X$)-LEN(Y$)
```