A fundamental constant

***Boris Allan** presents a program to calculate the value of e*

The exponential, *e*, the base of natural logarithms, is a fundamental constant in mathematics. *e* appears in many guises, especially in calculus, and has a very simple definition:

$e = 1 + 1/1! + 1/2! + 1/3! + 1/4! + 1/5! + \dots$

and onwards. In English, this means that *e* is the sum of 1 plus the reciprocal of the factorial of 1, plus the reciprocal of the factorial of 2, plus the reciprocal of the factorial of 3, and so forth. The factorial of, say, 4 is $4 \times 3 \times 2 \times 2 \times 1$ and so it is simple to realise that $4! = 4 \times 3!$

Rewriting the expansion for *e*, we can produce:

$e = 1 + t(1) + t(2) + t(3) + t(4) + t(5) + \dots$

where, in general, $t(n) = t(n-1)/n$ — think about it. To produce a value for *e* by a simple Basic program is not too difficult:

```
1000 M = 0 : T = 1 : E = 0
1010 REPEAT
1020 M = M + 1 : T = T/M : E = E + T
1030 UNTIL T = 0
```

and this will produce a value correct to about 7 decimal places — hardly an earth-shattering calculation, but then hardly earth-shattering accuracy either. I want to rule the digits.

Suppose I asked you to calculate the value of *e* correct to a small number of decimal places, say 20? Suppose that I was cruel (am cruel) and you had to work it out by hand (assume that you can still remember how to add, subtract, divide and multiply)? First, you would calculate term 1, ie *t(1)*, to 20 decimal places — actually the answer is 1. Second, you would take the result of *t(1)*, and divide that by 2, to obtain *t(2)* — answer .5. *t(3)* is .5 divided by 3, and does not finish exactly — .1 followed by an infinite number of 6s.

There are various ways in which it is possible to cope with these never-ending 6s, including: forget about it, just have a 1 and then nineteen 6s; round the last 6 up to give a 7; or perform the calculation to a greater number of digits, and forget about rounding until the very end (the end comes when all the digits of the term are zero). I propose to use the third variant, because it is simpler and is more accurate than rounding at the end of each term's 20 digits.

So, we have worked out how to perform the operation: take each succeeding term, divide it by the correct factor, and then add the result to the cumulating total. All we have to do is to work out how to divide long numbers (lots of little divisions), and then how to add all the terms together (and round the result).

The most important part of the program is that between lines 10 and 100, and in line 110, the formatter (@%) is set to produce 1 digit wide output (for the output at lines 300 to 330). At 120 the number of digits (*N%*) is input, and 4 added to that number — the extra digits accuracy. *Term%*(*N%*) is the array I use to store the *N%* digits of the successive terms, and *E%*(*N%*) is the array used to store the cumulated total (remember *N%* is now 4 more than the number of digits needed in the result).

Line 140 initialises the zero elements of *Term%* and *E%* to 1: the zero element is the whole number, and elements 1 to *N%* are the *N%* decimal places; the initial value of *e* (before any term) is 1, and the first term is also 1. *M%* (the term number)

```
   10REM--------------------------------
   20
   40REM  CALCULATION OF THE CONSTANT e
   50
   60
   70REM  (c) BORIS ALLAN, 1983
   90
  100REM--------------------------------
  110 @%=1 : REM SETS FORMAT
  120 INPUT N% : N%=N%+4
  130 DIM TERM%(N%),E%(N%)
  140 TERM%(0)=1 :E%(0)=1
  150 M%=1
  160
  170 REPEAT SUM%=0 : REM BUILDS UP TERMS
  180 FOR I%=0 TO N%
  190 IF TERM%(I%)<>0 THEN PROC_DIVISION
  200 NEXT I%
  210 M%=M%+1
  220 UNTIL SUM%=0 : REM ALL TERMS ARE ZERO
  230
  240 FOR I%=N% TO 1 STEP -1 : REM ISOLATES ENTRIES
  250 IF I%=N%-4 THEN E%(I%)=E%(I%)+E%(I%+1) DIV 5 : REM ROUNDING
  260 FOR J%=3 TO 1 STEP -1
  270 IF I%>=J% THEN PROC_SPLITTING
  280 NEXT J% : NEXT I%
  290
  300 N%=N%-4 : REM READY TO PRINT OUT RESULT
  305 VDU 2
  310 PRINT E%(0);".";
  320 FOR I%=1 TO N% : PRINT E%(I%); : NEXT I% : PRINT
  325 VDU 3
  330 END : REM OF MAIN PROGRAM
  340
  350 DEF PROC_DIVISION : REM DIVIDING BY M%
  360 IT%=TERM%(I%) DIV M%
  370 IF I%<>N% THEN PROC_TERMS
  380 ENDPROC : REM _DIVISION
  390
  400 DEF PROC_TERMS : REM ADDING TERMS
  410 TERM%(I%+1) = TERM%(I%+1) + 10*(TERM%(I%)-M%*IT%)
  420 TERM%(I%) = IT%
  430 E%(I%) = E%(I%)+IT%
  440 SUM%=SUM%+IT%
  450 ENDPROC : REM _TERMS
  460
  470 DEF PROC_SPLITTING : REM DISTRIBUTING TERMS
  480 SUM%=1 : FOR K%=1 TO J% : SUM% = SUM%*10 : NEXT K%
  490 IT%=E%(I%) DIV SUM%
  500 E%(I%-J%)=E%(I%-J%) + IT%
  510 E%(I%)=E%(I%) - SUM%*IT%
  520 ENDPROC : REM _SPLITTING
```

starts at 1 — line 150.

From 170 to 220 a short routine is repeated until a variable *Sum%* is zero: the first thing to happen is that *Sum%* is set to zero. For each element/digit from 0 to *N%* (line 180) a check is made to see if that element of the *Term%* array is zero, if not *Proc_division* is called.

The routine at 350 to 380 is made into a *Proc*, and not incorporated into the main program, because it clarifies the conditional statement at line 190 (and obviates the need for a *Goto*). At line 360, the existing value stored in the *I%*th element of *Term%* is integer divided by *M%* (the number of the term in the sequence). If the element is not the last in the array (ie *I%<>N%*) then a call is made to *Proc_terms* (again to save a *Goto*).

When one divides a number, the remainder of the division is carried on to the next digit in the sequence: this is what occurs in lines 410 to 420. Line 430 is where the array *E%* is cumulated. In line 440 *Sum%* is incremented and, if no arithmetic is performed (ie, all zeros), *Sum%* ends up as zero. After these calls, control returns to line 200 where *M%* is incremented by 1. This continues until *Sum%*=0, stasis.

The next segment (240 to 280) examines successive elements of *E%*, from the least significant leftwards. Lines 260 to 280 operate on three elements at a time, based on the element *I%* by use of *Proc_splitting*.

In line 480, *Sum%* takes the value 10, 100, or 1000, depending on the value of *J%* — this routine splits each value in an element steadily into digits (tens, hundreds and thousands), to take into account the fact that a value stored in *E%*(*I%*) will probably be greater than 9. The number of elements is greater (by 4) than the number of digits accuracy, and at line 250 the value stored in element *N%−4* is rounded.

The section from 300 to 330 prints out the value to the specified number of digits: lines 305 and 325 switch the printer on and off — you know what to do if you have no printer.

Here is a problem: Improve this routine, and implement it for positive and negative values of *X*, where the exponent of *X* is:

exp(X) = 1 + X/1! + X^2/2! + X^3/3! + . . .

First prize: 1 copy of my BBC book from Sunshine. Second prize: 1 copy of my BBC book from Sunshine, plus an autographed photograph. ■

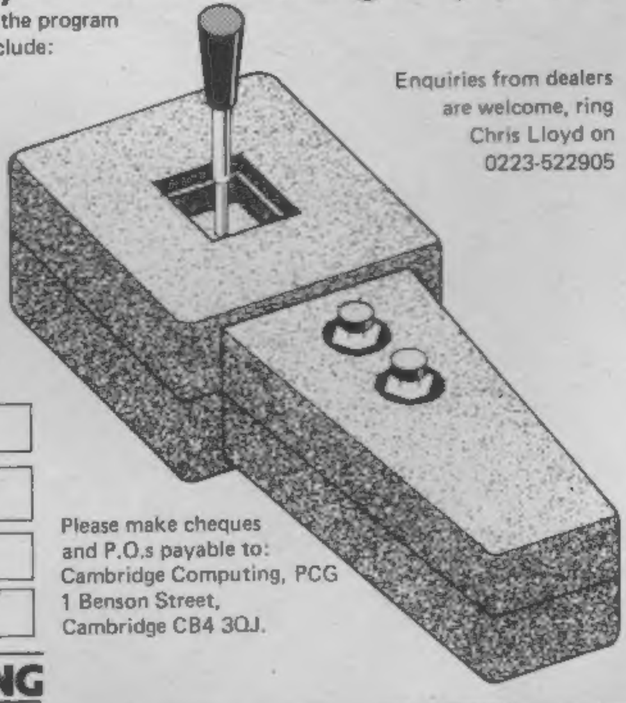
```
2.718281828459045235360287471352662497757247093699959574966967627724076630353547
59457138217852516642742746639193200305992181741359662904357290033429526059563073
81323286279434907632338298807531952510190115738341879307021540891499348841675092
44761460668082264800168477411853742345442437107539077744992069551702761838606261
33138458300075204493382656029760673711320070932870912744374704723069697720931014
16928368190255151086574637721112523897844250569536967707854499699679468644549059
87931636889230098793127736178215424999229576351482208269895193668033182528869398
49646510582093923982948879332036250944311730123819706841614039701983767932068328
23764648042953118023287825098194558153017567173613320698112509961818815930416903
51598888519345807273866738589422879228499892086805825749279610484198444363463244
96848756023362482704197862320900216099023530436994184914631409343173814364054625
31520961836908887070167683964243781405927145635490613031072085103837505101157477
041718986106873969655212671546889570350354
```

At last! A joystick that works!

Cambridge Computing bring you the first intelligent joystick.

Works on all existing software - regardless of which keys the program uses. No need for specially written software. Features include:

- Compatible with Spectrum, ZX81, Jupiter Ace
- 2 Independent Fire Buttons
- 8 Directional Microswitched action
- Plugs into edge connector
- Interface complete with edge connector
- Atari joystick compatible
- Joystick with Interface £29.90



Enquiries from dealers are welcome, ring Chris Lloyd on 0223-522905

Name..

Address...

....

Please send me:

..........joystick, interface, and tape @ £29.90.... £

Spectrum☐ ZX81☐ Jupiter Ace☐

..........interface and tape @ £24.00.................... £

Spectrum☐ ZX81☐ Jupiter Ace☐

..........joysticks @ £7.00.. £

Spectrum☐ ZX81☐ Jupiter Ace☐

Total including VAT.......... £

Please make cheques and P.O.s payable to: Cambridge Computing, PCG 1 Benson Street, Cambridge CB4 3QJ.

CAMBRIDGE COMPUTING