MULTIE – Multiprecision Euler Constant ***e***

Abstract

MULTIE is a simple program written in 1980 for the SHARP PC-1211 and PC-1212 pocket computers and compatibles to compute up to 572 digits of Euler's constant e. Two sample runs are included.

Keywords: *Euler constant e, multiprecision, SHARP pocket computer, PC-1211, PC-1212, compatibles*

1. Introduction

MULTIE is a simple 6-line *(~ 260 bytes)* practice program I wrote in 1980 for the *SHARP PC-1211* and *PC-1212* pocket computers and compatible models, which can compute up to 572 digits of *Euler*'s constant *e (=2.71828+)* using multiprecision arithmetic. It computes *e* using the formula:

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots$$

2. Program Listing

```
1: "A" CLEAR: INPUT "GR.7=";A: G=5E6,A(7+A)=G,B=2G,F=A
2: FOR D=3 TO 999: C=A(7+F): IF C=0 LET F=F+1,C=A(7+F):IF F=2A GOTO 6
3: FOR E=7+F TO 6+2A: A(E)=INT(C/D,C=A(E+1)+B*(C-DA(E): NEXT E
4: FOR E=6+A TO 7+F-A STEP -1: A(E)=A(E)+A(E+A): IF A(E)>=B LET A(E)=A(E)-B,A(E-1)=A(E-1)+1
5: NEXT E: NEXT D
6: USING "##.#######": BEEP 2: G=G+2B: FOR E=7 TO 6+A: PRINT A(E)/B: NEXT E
```

Notes:

- the **E** in G=5**E**6 at *line 1* is entered by pressing the [***Exp***] key *(Exponent)*, not the [E] key *(letter E)*.
- if converting to other *BASIC* dialects, remember that ***A*** maps to ***A(1)***, ***B*** maps to ***A(2)***,...***G*** maps to ***A(7),*** etc.
- also, take into account that implied multiplication is used throughout and the *automatic parentheses closing* feature is used at *line 3* to omit two final parentheses, saving memory.

3. Usage Instructions

See the worked examples to understand how to use the program.

4. Examples

The following examples can be used to check that the program was correctly entered and to understand its usage.

4.1 Example 1

Compute at least *70* correct digits of e.

In **DEF** Mode, proceed as follows:

```
SHFT A  →  GR. 7=_   (70 digits = 10 groups of 7 digits, plus one extra group to ensure we get at least 70 digits)
11 ENTER → 2.7182818 ENTER → 0.2845904 ENTER → 0.5235360 ENTER → 0.2874713
   ENTER → 0.5266249 ENTER → 0.7757247 ENTER → 0.0936999 ENTER → 0.5957496
   ENTER → 0.6967627 ENTER → 0.7240766 ENTER → 0.3035328 ENTER → >
```

The correct last block is **3035354** so we actually got *76 correct digits*, namely:

e *= 2.7182818 2845904 5235360 2874713 5266249 7757247 0936999 5957496 6967627 7240766 30353*

4.2 Example 2

Compute the full possible *572* digits of ***e***. *(it will take a while ...)*

In **DEF** Mode, proceed as follows:

```
SHFT A → GR. 7=_   (572 digits = 82 groups of 7 digits, which is the maximum for the available memory)
82 ENTER → 2.7182818 ENTER → 0.2845904 ENTER → ... ENTER → 0.8209245 ENTER → >
```

The correct last block is **8209392** so we got *572 correct digits*, namely:

e = 2.7182818	2845904	5235360	2874713	5266249	7757247	0936999	5957496	6967627	7240766
3035354	7594571	3821785	2516642	7427466	3919320	0305992	1817413	5966290	4357290
0334295	2605956	3073813	2328627	9434907	6323382	9880753	1952510	1901157	3834187
9307021	5408914	9934884	1675092	4476146	0668082	2648001	6847741	1853742	3454424
3710753	9077744	9920695	5170276	1838606	2613313	8458300	0752044	9338265	6029760
6737113	2007093	2870912	7443747	0472306	9697720	9310141	6928368	1902551	5108657
4637721	1125238	9784425	0569536	9677078	5449969	9679468	6445490	5987931	6368892
3009879	3127736	1782154	2499922	9576351	4822082	6989519	3668033	1825288	6939849
6465105	8209								

Notes

1. The program considers the multiprecision ongoing result as subdivided into *7-digit blocks*, 1 block per element of array *A*. Each 7-digit block is divided by *D* (2,3,...,*999*) and then remainders are propagated.

2. Each block is 7-digit long so as to allow for divisors *up to 3 digits*, i.e. up to *D=999*, memory permitting. Adding a 7-digit number to a 3-digit remainder times 10^7 gives a 10-digit result at most, so overflow is avoided. If *D* were restricted to 2 digits, i.e.: up *to D=99*, then 8-digit blocks could be used instead and both memory usage and computation time would be reduced but 572 digits wouldn't be achievable as computing that many digits requires using divisors up to *D=285.*

3. In order to show intermediate *0's* correctly, each 7-digit block (but the first) is divided by $\mathbf{10^7}$ before outputting it (thus rendering it as *0.ddddddd* where all the *d's* are significant and can be ***0***), lest the leftmost digits of a block would not be output if zero. Exceptionally, the very first block is output as *2.ddddddd.*

Copyrights