

Fakultät - Wo sind die Grenzen ?

(? des Computers und/oder des Programmierers ?)

(etwas über Computer-Unendlichkeit und Computer-Null)

Dr. Ralf Lemke Mont-Cenis-Str. 567 44627 Herne Tel. 0 23 23 - 6 81 49

Die Fakultät vom "WINDOWS Rechenknecht"

Ein DOS-PC mit WINDOWS besitzt einen Rechner für technisch-wissenschaftliche Anwendungen. Mit der Taste (n!) lassen sich Fakultäten berechnen, die z.B. für Naturwissenschaftler wichtig sind. Auch Statistiker, die eine Wahrscheinlichkeit von Lottogewinnen ausrechnen, benutzen Fakultäten.

Und was ist die Fakultät n! ?

n! ist das Produkt aller Zahlen von 1 bis n (1 mal 2 mal 3 mal ... n).

Probiert man das aus, dann erhält man $15! = 1307674368000$. $16!$ wird schon als eine mit einer Zehnerpotenz zu multiplizierende Zahl angezeigt ($2,09...e+013$). Die größte Fakultät, die der Rechner noch "schafft", ist $170! = 7,25 e+306$.

Es ist aber recht einfach, weiter zu rechnen als der Rechenknecht von WINDOWS!

Ein Basic-Programm, so gut wie der WINDOWS-Rechner

In "Q-Basic" können wir Fakultäten als eine FOR ... NEXT - Schleife programmieren. Dabei verwenden wir doppelte Genauigkeit, damit unser Versuch mit dem "WINDOWS-Rechner" mithalten kann.

!!! etwa hier "Listing FAKULT00"

Wir erhalten $18! = 6402373705728000$ und $170! = 7,25...D+306$. Fakultäten, die über $170!$ hinausgehen, schafft unser kleines Programm nicht mehr; der Rechner meldet einen "ÜBERLAUF". Sollen wir deshalb aufgeben?

Das, was ein einfaches Programm nicht schafft, ist möglicherweise kompliziert, aber unlösbar muß das Problemchen deshalb nicht sein. Also versuchen wir es mal mit den kleinen grauen Zellen (und nicht wie bei einem Überlauf in der Waschküche mit "abpumpen").

Warum, so fragen wir uns, hört der Rechner auf zu rechnen?

Will er nicht mehr? kann er nicht mehr? darf er nicht mehr?

Die Antwort ist : er darf nicht, weil er nicht mehr kann! und er darf nicht wollen, weil es ihm verboten wurde! Die Fehlermeldung "ÜBERLAUF" soll das andeuten - wir wollen versuchen das zu erklären.

Was ist Unendlich?

Für eine Badewanne und für den mathematischen Prozessor bedeutet "ÜBERLAUF", daß der Topf so voll ist, daß kein weiterer Tropfen und keine weitere EINS zusätzlich hineinpassen. Überlauf paßt zur Badewanne und da kann das Wasser ablaufen oder abgepumpt werden; für den Prozessor wollen wir einen anderen Ausdruck benutzen, der nur auf den ersten Blick nicht richtig zu passen scheint: "ÜBERLAUF" ist das Erreichen der numerischen Unendlichkeit für den Prozessor.

Ein "ÜBERLAUF für den Rechner" ist einfach zu erklären, denn bis zu einer vorgegebenen großen Zahl (ÜBERLAUF-minus-EINS) geht (fast) jede Rechnung, bei "ÜBERLAUF (= ZWEI-hoch-1024)" und mehr geht gar nichts mehr.

"Unendlichkeit" im mathematischen und im menschlichen Sinne ist dagegen immer noch sehr weit weg von jedem denkbaren Ergebnis, gleichgültig wie groß das Ergebnis auch sein mag.

Wie vermeidet man die Unendlichkeit?

Damit wir den Überlauf nicht erreichen, können wir auch (1 durch HUNDERT mal 2 durch HUNDERT mal 3 durch HUNDERT mal ... n durch HUNDERT) rechnen. Und schon bekommen wir $171! = 1,2415180702176$, allerdings mit einer noch nicht richtigen Zehnerpotenz. Auch für $722!$ liefert uns diese Methode ein Ergebnis, bei dem ebenfalls die Zehnerpotenz zu berichtigen ist. Die fehlende Zehnerpotenz läßt sich einfach ermitteln.

Wir benutzen zuerst eine ganz gut brauchbare Methode. Unser Ergebnis ist so groß oder so klein, daß wir zur Zehnerpotenz vom Rechenergebnis einfach noch ZWEI-mal-Fakultät-(ohne-!) addieren.

!!! etwa hier "Listing FAKULT01"

Unsere "geteilt durch HUNDERT" Schleife sorgt dafür, daß das Zwischenprodukt zunächst beständig kleiner wird. Wenn die Schleife mit HUNDERT-durch-HUNDERT fortgesetzt wird, dann wird das Zwischenprodukt langsam wieder größer, bis es bei $723!$ zum "ÜBERLAUF" kommt. Dazwischen haben wir von $266!$ bis $303!$ Ergebnisse, die von unserem Programm ohne Zehnerpotenz angezeigt werden und die deshalb mit der zunächst brauchbaren Methode zu falschen Angaben führen. Deshalb müssen wir entweder bei $265!$ aufhören, oder nach einer geeigneteren Methode zur Bestimmung der Zehnerpotenz suchen.

!!! etwa hier Graphik "Der Weg zur UNENDLICHKEIT mit FAKULT01"

Über die NULL und wie ein PC die NULL beherrscht!

Was mit dem Divisor HUNDERT geht, das sollte auch mit dem Divisor TAUSEND gehen.

Ich habe das probiert. Aber das ging nur gut bis $380!$.

$381!$ geht nicht mehr, denn da streikt das Programm mit einem reziproken ÜBERLAUF (entsprechend einer NULL).

Für uns ist es eine klare Sache: NULL ist 0. Nicht so für den Prozessor des Rechners. Der rechnet (weitgehend) ordentlich bis zu einer vorgegebenen kleinen Zahl und alles was nur ganz geringfügig kleiner ist, das betrachtet der Rechner als NULL. So einfach geht es also noch nicht weiter.

Aber mit etwas Überlegung geht es doch. Wir fangen einfach nicht mit EINS-durch-TAUSEND-mal- usw. an, sondern z.B. bei HUNDERT-durch-TAUSEND. Die Reichweite des Programms wird dadurch bis zu $2362!$ erweitert. Bis hierher funktioniert dann die Berechnung der Zehnerpotenz mit der brauchbaren Methode.

!!! etwa hier "Listing FAKULT10"

Damit kommen wir zum nächsten Problem, an das man eigentlich nicht denkt, wenn es um die Fakultät geht. Und noch einmal hängt es am mathematischen Schaltzentrum des PC.

Menschen rechnen im Dezimalsystem und ein Computer rechnet intern im Dualsystem. Wir runden ein "K" des Rechners auf etwa 1000, der Computer macht aus einer Zehnerpotenz (die wir an den Rechner übergeben) eine fast perfekte Potenz von ZWEI - und meistens, aber nicht immer - bemerken wir keinen Unterschied im Ergebnis, das uns der Rechner liefert. (Der Pentiumfehler war so ein BUG, der übrigens nicht auf die verbesserten Prozessoren beschränkt war!)

Bei unserer Fakultät häufen sich kleine Fehler des rundenden Prozessors in den Zwischenprodukten.

Das bemerken wir möglicherweise gar nicht, wir vermuten das ja auch zunächst nicht. Die "numerische Ungenauigkeit" von Reihenrechnungen mit Computern als Folge von Rundungsfehlern ist Fachleuten allerdings lange bekannt.

Die Rundungsfehler werden umso gravierender, je näher ein Ergebnis (oder ein Zwischenergebnis) in die Nähe des "erlaubten Zahlenbereichs" kommt.

Wir haben Rundungsfehler nahe an der Grenze des erlaubten Zahlenbereiches, zusammen mit der Fehlersummierung durch Reihenrechnung, danach braucht man nur zu suchen und findet sicher auch Beispiele ($999!$ und $1000!$).

FAKULT10 ist praktisch zu verbessern, wenn man anstelle von "mal-0,001" ein "geteilt-durch-1000" verwendet(!). Ein kleiner Unterschied mit beträchtlicher Wirkung! (Das gab es schon beim CompuCorp322G, beim HP65, beim TI59 und beim SR52!)

Eine Erweiterung der berechenbaren Fakultäten bis 3110! liefert der "Einstieg" in die "geteilt-durch-Tausend-Routine" bei 170 und eine zweckmäßigere Ermittlung des Exponenten über den Zehnerlogarithmus des Endproduktes. Jetzt kann das Programm durch das "Tal, nahe bei der NULL" bis fast zum "Gipfel nahe bei UNENDLICH" rechnen. Abweichungen durch Rundung und/oder Reihenrechnung sind hier nicht mehr direkt festzustellen.

!!! etwa hier "Listing FAKULT11"

!!! etwa hier Graphik "Ergebnisse zwischen NULL und UNENDLICH mit FAKULT11"

Fakultät von noch größeren Zahlen

Fakultäten großer Zahlen berechnet man entweder näherungsweise oder mit hier nicht erwogenen "langen Zahlen" (Erweiterung des Zahlenbereichs, das ist extrem zeitaufwendig und man benötigt eventuell mehrere Bildschirme, um das Ergebnis darzustellen). Meist wird mit

!!! etwa hier "STIRLING 1" $n! \approx \left(\frac{n}{\exp}\right)^n \cdot \sqrt{2n \cdot \pi} ; n = 1, 2, 3, \dots$

gerechnet. Diese Näherung ist für große Werte ($n > 10^8$) auf ca. 15 Stellen genau. Es handelt es sich um eine abgespeckte Version der Formel nach STIRLING.

!!! etwa hier "STIRLING 2" $n! = \left(\frac{n}{\exp}\right)^n \cdot \sqrt{2n \cdot \pi} + h ; \left[0 < \frac{h}{n!} < \frac{1}{12n}\right]$

Deutlich besser als STIRLING für Berechnungen im Bereich von 100! bis (10-hoch-8)! ist eine andere Näherung, die auf einer Reihenentwicklung nach EULER und McLAURIN unter Verwendung von BERNOULLI-Koeffizienten beruht. Damit erreichen wir eine Genauigkeit von mindestens 14 geltenden Stellen, allerdings bezieht sich die Zuverlässigkeit auf die Summe der Stellenzahlen von Basis und Exponent, wobei der Exponent zuerst zu berücksichtigen ist (für STIRLING gilt das ebenfalls).

!!! etwa hier "BERNOULLI" $\ln(n!) = (n + 0,5) \cdot \ln(n) - n + \frac{1}{12n} - \frac{1}{360n^3} + \frac{1}{1260n^5} - \frac{1}{1680n^7} + \frac{\ln(2\pi)}{2}$

Fragen sind nicht indiskret - Antworten sind es schon mal - oder Schuld ist der Programmierer und nicht der Anwender

Für ein ordentliches Programm gehört es sich, daß denkbare Bedienungsfehler abgefangen werden. Die Fakultät ist nur für ganze und positive Zahlen definiert. Deshalb sollten negative Eingaben genauso wie nicht ganzzahlige Eingaben von einer Berechnung ausgeschlossen werden. Es sollte hier erwähnt werden, daß viele hier nicht zulässige Eingaben zum Teil über eine "Fakultäts-Ober-Funktion", die sogenannte Gamma-Funktion, definierte Ergebnisse besitzen.

!!! etwa hier "Gamma Funktion" $\Gamma(n) = (n-1)! \quad n > 0; \text{ ganzzahlig}$
 $= \int_0^\infty x^{n-1} \exp^{-x} dx \quad n \neq 0, -1, -2, \dots$

Wenn aus Versehen ein Buchstabe anstelle einer Zahl eingegeben wird, dann sollte das Programm das natürlich auch feststellen und eine entsprechende Fehlermeldung ausgeben.

An der oberen Grenze unserer Fakultätsberechnung wird angezeigt, in wie genau das Ergebnis ist.

Es fehlt eine Routine, die das dezimale Komma im Zehnerblock als programminterne Fehlermeldung abfängt, aber das erledigt Q-BASIC für uns (mehr schlecht als recht und nötig ist es nicht unbedingt). Eventuell ließe sich das durch eine das Programm startende BATCH-Datei erledigen – hier ist das nicht versucht worden.

Was nutzen uns so großen Fakultäten?

Wenn wir jetzt sehr große Fakultäten berechnen können, dann haben wir nebenher in mehrfacher Hinsicht profitiert:

Wir können UNENDLICHKEIT und NULL als numerische Grenzen des Rechners verstehen und berücksichtigen. Außerdem haben wir gelernt, daß es für den PC ein Unterschied ist, ob wir mit 0,001 multiplizieren oder durch 1000 dividieren, dabei sind uns auch die kumulierten Rundungsfehler bei Reihenrechnungen begegnet. Das ist sicher nützlich, auch wenn die ganz großen Fakultäten von nur sehr wenigen Spezialisten benötigt werden.

- [1] Handbook of Tables for Mathematics, S. 48, 3.Aufl. 1971, CRC & Co, Cleveland, Ohio.
- [2] Handbook of Tables for Mathematics, S. 592, 3.Aufl. 1971, CRC & Co, Cleveland, Ohio.
- [3] R. Lemke, Der "Plus-Punkt", ein Fehler im Betriebssystem und seine Vermeidung bei Rundungen 64'er, 10/1989, 72
- [4] R.Lemke, prämiertes 20- Zeiler zur Berechnung von Fakultäten $< 10^{15}$, 64'er, 11/1991, 38

Begleittexte zu den Listings

Listing 00 FAKULT 00
Ein Basic-Programm, das so viel kann wie der WINDOWS-Rechner

Listing 01 Fakult 01
Das ist die Idee. Guter Anfang, nicht 0/8/15, aber die Ausführung ist noch nicht perfekt.
Im Programmzeile 100 sollte man das erste GOTO 20 entfernen, um Fehler in der Anzeige zu finden.

Listing 10 Fakult 10
Das ist die weitergehende Idee. Aber 999! und 1000! haben noch "eine Macke".

Listing 11 Fakult 11
Es wird immer besser. Nicht nur die richtige Rechnung ist jetzt programmiert, es geht auch noch ein wenig weiter in die hohen Zahlen.

Fakultät Fakultät
ist das Programm, das ich verwende und eventuell bei 3000! nicht alle berechneten Stellen benötige. Bei noch größeren Fakultäten sorgt die programmierte Rundungsroutine für die mit den Bernoulli-Koeffizienten erzielbare Zuverlässigkeit in den angegebenen Stellen.

$$\begin{aligned}
 99! &= 9,332\ 621\ 544\ 394\ 41 \times 10^{155} \\
 100! &= 9,332\ 621\ 544\ 394\ 41 \times 10^{157} \\
 999! &= 4,023\ 0872\ 600\ 770\ 902 \times 10^{2\ 564} \\
 10^3! &= 4,023\ 0872\ 600\ 770\ 902 \times 10^{2\ 567} \\
 99999! &= 2,824\ 229 \times 10^{456\ 568} \\
 10^5! &= 2,824\ 229 \times 10^{456\ 573} \\
 9999999! &= 1,202\ 4 \times 10^{65\ 657\ 052} \\
 10^7! &= 1,202\ 4 \times 10^{65\ 657\ 059} \\
 999999999! &= 9,90 \times 10^{8\ 565\ 705\ 513} \\
 10^9! &= 9,90 \times 10^{8\ 565\ 705\ 522} \\
 9999999999! &= 2,3 \times 10^{95\ 657\ 055\ 176} \\
 10^{10}! &= 2,3 \times 10^{95\ 657\ 055\ 186} \\
 10433891463! &= 9,9 \times 10^{999\ 999\ 999\ 993} \\
 10433891464! &= „unendlich“ für das Betriebssystem und die verwendete Näherungsmethode; Ergebnis ist $> 10^{999\ 999\ 999\ 999}$$$

dieser Wert ist wie alle weiteren quasi sofort verfügbar.
spätere Versionen von Windows brauchen dafür sehr viel Zeit.

FAKULT00

```

20 : PRINT
   : PRINT " Eingabe von n ";
   : INPUT N$
   IF N$ = "" THEN END

30  NN# = VAL(N$): N# = INT(NN#)
   IF N# <> NN# OR N# < 0 GOTO 10

50  fac# = 1
   FOR i = 1 TO N#
     fac# = CDBL(fac# * i / trick)
   NEXT

70  : PRINT SPC(6 - LEN(STR$(N#))); N#, "! = ";
   : PRINT fac#
   : PRINT
   :: GOTO 20

```

FAKULT01

```

   : PRINT
   : GOTO 20

10 : PRINT , , "? das geht nicht !"
   : PRINT

20 : PRINT
   : PRINT " Eingabe von n ";
   : INPUT N$
   IF N$ = "" THEN END

30  NN# = VAL(N$): N# = INT(NN#)
   IF N# <> NN# OR N# < 0 GOTO 10
   IF N# > 789 GOTO 10

50  fac# = 1: trick = 1
   IF N# > 170 THEN trick = 100
   FOR i = 1 TO N#
     fac# = CDBL(fac# * i / trick)
   NEXT
   IF N# > 170 GOTO 100

70  : PRINT SPC(30 - LEN(STR$(N#))); N#, "! = ";
   : PRINT fac#
   :: GOTO 20

100 : PRINT SPC(21 - LEN(STR$(N#))); N#, "! = ";
   :: PRINT fac#, "* 10 hoch"; 2 * N#
   :: GOTO 20
   : PRINT LEFT$(STR$(fac#), 17); " E";
   P$ = RIGHT$(STR$(fac#), 4)
   IF LEFT$(P$, 1) = "D" THEN P$ = RIGHT$(P$, 3)
   : PRINT VAL(P$) + N# * 2
   :: GOTO 20

```

FAKULT10

```
: PRINT
:: GOTO 20

10 : PRINT , , "? das geht nicht !"
   : PRINT

20 : PRINT
   : PRINT " Eingabe von n ";
   : INPUT N$
   IF N$ = "" THEN END

30  NN# = VAL(N$): N# = INT(NN#)
   IF N# <> NN# OR N# < 0 GOTO 10
   IF N# > 2362 THEN PRINT , , " ? zu gross !": GOTO 10

50  fac# = 1: ii = 1: trick = 1
   IF N# > 170 THEN fac# = 9.332621544394415#: ii = 100: trick = .001
   FOR i = ii TO N#
     fac# = CDBL(fac# * i * trick)
   NEXT
   IF N# > 170 GOTO 100

70 : PRINT SPC(30 - LEN(STR$(N#))); N#; "! = ";
   : PRINT fac#
   :: GOTO 20

100 P$ = RIGHT$(STR$(fac#), 4)
    IF LEFT$(P$, 1) = "D" THEN P$ = RIGHT$(P$, 3)
    : PRINT SPC(43 - LEN(STR$(N#))); N#; "! = ";
    : PRINT LEFT$(STR$(fac#), 17);
    : PRINT " * 10 "; CHR$(24); VAL(P$) + 3 * N# - 142
    :: GOTO 20
```

FAKULT11

```

: PRINT
:: GOTO 20

10 : PRINT , , "? das geht nicht !"
   : PRINT

20 : PRINT
   : PRINT " Eingabe von n ";
   : INPUT n$
   IF n$ = "" THEN END

30  Nn# = VAL(n$); n# = INT(Nn#)
   IF n# > 3110 GOTO 200
   IF n# <> Nn# THEN PRINT , "nur mit Gamma-Funktion !": GOTO 10
   IF n# < 0 GOTO 10
   IF LEFT$(n$, 1) < "0" OR LEFT$(n$, 1) > "9" GOTO 10

50  fac# = 1: trick = 1: ii = 1
   IF n# > 170 THEN fac# = 7.257415615307994D-80
   IF n# > 170 THEN trick = 1000: ii = 171
   FOR i = ii TO n#
     fac# = CDBL(fac# * i / trick)
   NEXT
   IF n# > 170 GOTO 100

70 : PRINT SPC(30 - LEN(STR$(n#))); n#; "! = ";
   : PRINT fac#
   :: GOTO 20

100 Ifac = INT(LOG(fac#) / LOG(10))
   : PRINT SPC(43 - LEN(STR$(n#)));
   : PRINT n#; "! = "; STR$(fac# / 10 ^ Ifac);
   : PRINT " * 10 "; CHR$(24); Ifac + n# * 3 - 124
   :: GOTO 20

200
   IF n# > 10433891463# THEN PRINT , , " zu wenig genau": GOTO 20
   IF n# > 1723507# THEN REM noch gut 4 geltende Stellen
   IF n# > 14842906# THEN REM noch gut 3 geltende Stellen
   IF n# > 130202808# THEN REM noch gut 2 geltende Stellen
   IF n# > 1158787577# THEN PRINT , , "noch 2 geltende Stellen"
     X# = CDBL(.9189385332046728# + (n# + .5) * LOG(n#))
     X# = CDBL(X# - 1 / 360 / n# ^ 3 + 1 / 1260 / n# ^ 5)
     X# = CDBL(X# - 1 / 1680 / n# ^ 7 - n# + 1 / 12 / n#)
     LX# = CDBL(X# / LOG(10))
     P# = INT(LX#): P = LEN(STR$(P#))
     A# = CDBL(10 ^ (LX# - P#))
   : PRINT SPC(30 - LEN(STR$(n#))); n#; "! = ";
   : PRINT LEFT$(STR$(A# + .5 * 10 ^ (P - 14)), 16 - P);
   : PRINT " * 10 "; CHR$(24); P#
   :: GOTO 20
   REM nur Exponent & knapp eine Stelle bei N# > 94 851 898 540

300
' 25/26.Oktober 1992 Ralf L E M K E
' Mont-Cenis-Str.567 44627 H E R N E
' Telefon 0 23 23 6 81 49

```

FAKULTÄT.BAS LISTING mit Komfort

SCREEN 9: CLS : PRINT

PRINT , "am "; MID\$(DATE\$, 4, 3); LEFT\$(DATE\$, 3); RIGHT\$(DATE\$, 2); " um "; TIME\$
REM 25.-27.Oktober 1992 Ralf LEMKE, modifiziert10 COLOR 9, 0: PRINT : PLAY "o4 l9 a"
PRINT " Eingabe der Basis f r die Fakult,t "; : COLOR 15
PRINT , " -----"; CHR\$(26); , : INPUT n\$

```

nn# = VAL(n$): n# = INT(nn#): IF n$ = "" THEN END
IF n# <> nn# THEN COLOR 12, 1: PRINT SPC(5); "! so jeht dat nich "; CHR$(19); " dazu brauchze die
Gamma-Funktion !": PLAY "o4 L8 g. ee p7 f. dd p3": GOTO 10
IF LEFT$(n$, 1) > "9" OR LEFT$(n$, 1) < "0" OR n# < 0 THEN PRINT SPC(37); "! aber so nicht !": PLAY "mf
o2 L20 ed+ ed+ e o1 b o2 d c L4 o1 a p4": GOTO 10
IF n# > 3109 GOTO 200: REM wenn es sehr eilt, dann bei kleinerem n# GOTO 200 (auf Kosten der
Genauigkeit) !
IF n# > 170 GOTO 100
COLOR 10
fac# = 1
FOR i = 1 TO n#: fac# = CDBL(fac# * i): NEXT
PRINT SPC(9 - LEN(STR$(n#))); n#, "! ="; fac#
GOTO 10

```

```

100 COLOR 15
REM das kostet Zeit !
REM die Ergebnisse sind erstaunlich zuverl,,ssig (gute 14 Stellen ! zus,,tzlich zum Exponent !) !
fac# = 7.257415615307992D-80: trick = 1000
FOR i# = 171 TO n#: fac# = CDBL(i# / trick * fac#): NEXT
fac# = fac# * (1 + n# / 6.21E+16): REM mit diesem Faktor wird die 15. Stelle im Ergebnis noch
n,,herungsweise richtig !!
lfa = INT(LOG(fac#) / LOG(10))
PRINT SPC(16 - LEN(STR$(n#))); n#, "! ="; STR$(fac# / 10 ^ lfa); " * 10 "; CHR$(24); lfa + n# * 3 - 124
GOTO 10

```

```

200 COLOR 11
IF n# > 10433891463# THEN COLOR 12, 1: PRINT SPC(36); "liefert kein zuverl,,ssiges Ergebnis mehr !":
PLAY "mf o2 L20 ed+ ed+ e o1 b o2 dc L4 o1 a p4": GOTO 10
IF n# > 1723507# THEN COLOR 13: BEEP 'noch gut 4 geltende Stellen
IF n# > 14842906# THEN COLOR 10: PLAY "fg" 'noch gut 3 geltende Stellen
IF n# > 130202808# THEN COLOR 12: BEEP 'noch gut 2 geltende Stellen
IF n# > 1158787577# THEN COLOR 14: PLAY "o0 L9 abab p7": PRINT SPC(13); "nur noch gerade 2
geltende Stellen !"
ll# = CDBL(1 / 12 / n#)
x# = CDBL(.9189385332046727# + ll# - ll# / 30 / n# / n# + ll# / 105 / n# ^ 4 - ll# / 140 / n# ^ 6 + (n# + .5)
* LOG(n#)) - n#
lx# = CDBL(x# / LOG(10)): p# = INT(lx#): p = LEN(STR$(p#)): a# = CDBL(10 ^ (lx# - p#))
PRINT SPC(23 - LEN(STR$(n#))); n#, "! ="; LEFT$(STR$(a# + 5 * 10 ^ (p - 14)), 16 - p); " * 10 ";
CHR$(24); p#
GOTO 10

```

```

REM PRINT 4.023872600770938# / 4.023872600770838#, 4.149359603437854# / 4.149359603437652#
REM 999 ! mit derive 999 ! hier 3000 ! mit derive 3000 ! hier
REM = 1.0000000000000025# = 1.0000000000000049#
REM 4.0238726007709377354702434e2564 4.14935960343785408555686709e9130

```