ExCalc

COLLABORATORS			
	TITLE : ExCalc		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 8, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

ExCalc

1.1 ExCalc.guide

ExCalc

The Best Calculator For Any Computer

Version 1.0

SHAREWARE

1.2 Introduction

Introduction

ExCalc is the first calculator available for any computer which provides calculations which can be accurate to over 50 digits! As well all decimals are represented exactly so you never loose pennies during a calculation. All this power is made more accessible through the choice of a GUI interface or command line entry of complete equations -- not just numbers. Any entered equation can be recalled just by clicking on a history window. You also have access to sixteen memory locations which can store any number.

Other features include:

- o full scientific, transcendental, power, and logarithmic operations
- o programmer functions like and, or, shift, rotate, etc., on 172-bits!
- o scrolling history window of equations, memory, or results
- o choice of radian, degree, or gradian angle measurements
- o from 8 to 52 digits
- o numbers from -1E9999 to 1E9999
- o exact decimal number representation
- o convenience functions like square and cube roots, reciprocal, etc
- o storage of equations and memory as long as your computer is on
- o custom fonts with square root and cube root characters
- o complete calculator engine source code (in Oberon-2) is included

Registering will also get you the complete source code for the GUI which was written using CanDo. You will be able to customize the GUI, menus, and

keypad configuration any way you like -- provided you have CanDo.

1.3 System Requirements

System Requirements

This calculator should work on any Amiga running version 2.0 of the operating system or higher. The base calculator uses a calculation engine which is optimised for a 68000 processor. Other processing engines for 68020 and 68030 are available when you register. See below for more details.

You shouldn't have any problems running the calculator on a 512K Amiga -- although I haven't tried it. Who has a 512K Amiga anymore anyway?

1.4 Installation

Installation

To install the calculator just drag the complete ExCalc drawer over to wherever you want the calculator to reside on your hard disk or floppy diskette.

1.5 Quick Start

Quick Start

You can use the calculator right away. Just double-click on the ExCalc icon. Read on if you want to get some background information on how to use the calculator and want a brief tutorial.

1.6 Why Another Calculator?

Why Another Calculator?

The Amiga has an almost embarrassing abundance of floating point math libraries which include the Fast Floating Point (FFP) library, IEEE 32-bit and IEEE 64-bit floating point libraries. The latter two are supported by the Amiga's floating point coprocessor. But there are serious deficiencies in all of these numerical representations.

The chief problem with these floating point formats is that they are encoded as binary numbers (mathematical base 2) while we tend more naturally to decimal numbers (base 10). As a consequence, the fractional representations of the binary numbers are usually slightly different from the decimal number fractions

that we would expect. For example, the number 0.8 is exactly representable as a decimal number while the closest binary fraction, in IEEE 32-bit format, ends up being 0.799999952316284. This difference is very important to business people who can't get their spreadsheets to balance when accumulated round-off errors don't balance out to zero. ExNumbers can represent all decimal fractions exactly so no pennies are gained or lost.

A second problem is that the maximum resolution supported by the IEEE 64-bit format is only about 15 digits. This limitation may be very serious to very large corporations whose accumulated income is numbered in billions of dollars and to space explorers who need to send a space probe to an exact orbital position. With a virtually unlimited number of digits, ExNumbers can again meet the need for higher resolutions.

There is a down side to the use of ExNumbers. Since they are implemented in software, they are slower than coprocessor-based floating point numbers for an equivalent number of digits. Certainly, when using many digits, the calculations are slower. In the proposed calculator application, however, the slower speed is not noticeable.

1.7 ExNumber Structure

ExNumber Structure

ExNumbers are represented as an array of 16-bit signed words, each of which contains four decimal digits and is called a Quad. The exponent is kept in a separate 16-bit word; and the number's sign is an enumeration of 'positive' and 'negative' values.

ExNumber Quads are different from Binary-Coded Decimal (BCD) representations which are also used to store decimal-encoded numbers. The Quads actually encode four decimal digits using a binary number so they are stored more efficiently and are faster than BCD numbers in calculations.

For example, to encode 123456.789 as an ExNumber, we first need to normalize this number to a value whose mantissa is between -10 and 10. In other words, the number is represented in scientific notation as 1.23456789E6. Next, this number is broken into groups of four digits, beginning at the leftmost digit. The decomposed number can now be represented as 1234 5678 9000 E0006 where the E0006 is the number's exponent. Note the trailing zeros in the 9000 Quad. These extra digit place holders are required to pad this Quad because, if they were omitted, the final 9 would be interpreted as 0009 which would lead to the erroneous number 123456.780009. The three Quads from this example are stored in binary form as shown in Figure 1.

Quad Index	Quads	Binary (16-bit integer)
0	1.234	0000 0100 1101 0010
1	5678	0001 0110 0010 1110
2	9000	0010 0011 0010 1000
.		.
.		.
.		.

```
      12     0000       0000 0000 0000 0000
   Exponent   +5        0000 0000 0000 0101
     Sign  positive     0000 0000 0000 0000
```

Figure 1: ExNumber representation of the number ‘123456.789’

1.8 Utility Routines

Utility Routines

Before dealing with the main mathematical operations, I need to introduce a few utilities which will be used in these algorithms.

The first of these is the ExCompare function which produces a result to indicate whether one ExNumber is equal to, greater than, or less than another ExNumber. Although the algorithm in the ExNumbers module for ExCompare appears confusing, Table 1 summarizes the decisions which are used to produce an ExNumber comparison. The notation A(+) indicates that A is positive, Aexp is an abbreviation for A’s exponent, and A(i) represents the ith Quad of A.

A(+)	B(+)	A=B	A(i)<B(i)	Aexp>Bexp	Aexp=Bexp	Result
T	F	-	-	-	-	A>B
F	T	-	-	-	-	A<B
T	-	-	-	T	F	A>B
F	-	-	-	T	F	A<B
T	-	-	-	F	F	A<B
F	-	-	-	F	F	A>B
-	-	T	-	-	T	A=B
T	-	F	T	-	T	A<B
F	-	F	T	-	T	A>B
T	-	-	F	-	T	A>B
F	-	-	F	-	T	A<B

‘T’-True, ‘F’-False, ‘-’-Don’t Care

Table 1: ExNumber comparison algorithm in tabular form

The ExChgSign procedure negates a number which means that the sign is toggled from positive to negative and vice versa.

ExAbs takes the absolute value of a number by forcing the sign to be positive.

ExNorm normalizes an ExNumber by removing leading zeros in a fraction and adjusting the exponent to guarantee a mantissa between -10 and 10. For example, 0.0000456 would be normalized to 4.56E-6.

ExTimes10 increments the exponent by 1 to simulate a multiplication by 10. This procedure is much faster than really multiplying an ExNumber by 10.

Similarly, ExDiv10 subtracts 1 from the exponent to simulate a division by 10. This procedure is also much faster than attempting to divide an ExNumber by 10.

The ExShiftRight procedure is used for shifting a single digit rightmost into an ExNumber's mantissa. Shifting 8 into the number 6.7892 produces the number 8.67892.

ExShiftLeft shifts an ExNumber to the left by a single digit and replaces the least significant digit with a zero.

The IsZero function returns true if the ExNumber argument passed to it is equal to zero.

1.9 Basic Arithmetic

Basic Arithmetic

Some of us may remember how confusing it was when first learning about binary numbers after having been exposed for most of our lives to decimal numbers. Well, the bad news is that you can forget most of what you learned about binary arithmetic; the good news is that the basic ExNumber operations of addition, subtraction, multiplication, and division are performed in very much the same way that we are used to from our daily lives.

Addition is the most basic operation in the ExNumbers module and the ExAddUtility is the heart of the addition algorithm which is used by the exported ExAdd procedure. Two positive numbers are added together in three steps: first, set the exponent of the result to the exponent of the larger number; second, shift the smaller number so that its Quads are aligned with the larger number; and finally, add together all the related Quads. The example in Figure 2 illustrates how the numbers 1.2345678E9 and 3.21456E5 are added together.

```
                  Quads       Exponent
        x    1.234 5678 0000     +9
        y    0.000 3214 5600     +9
      ------------------------------
    x + y    1.234 8892 5600     +9
```

Figure 2: Added ExNumbers 1.2345678E9 and 3.21456E5

The ExSubUtility subtracts two numbers using almost the same steps also used by the ExAddUtility with the exception that Quads are subtracted from each other instead of being added.

To simplify both addition and subtraction algorithms, I made a tacit assumption that both numbers would be positive. The reason this works is seen in how the ExAdd procedure checks and manipulates the signs of the two numbers to be added together. There are two possibilities: both numbers have the same sign (either positive or negative) so they can be added together using the ExAddUtility procedure; or the numbers have different signs so we subtract the negative number from the positive number using the ExSubUtility procedure.

The ExSub procedure is even simpler. This algorithm makes use of the well-known property that B - C can be rewritten as B + (-C). Thus, by negating

C, a call to the ExAdd procedure produces the correct answer.

ExMult multiplies two ExNumbers together using the same techniques that we were taught in school. Two nested loops produce a product by using the outer loop to index the first number's ith Quad and then the inner loop multiplies this Quad by each of the Quads in the second number to produce an intermediate product. Any carries are then shifted into this intermediate product, the exponent is adjusted accordingly, and the intermediate product is added to the final result. The above process is repeated until an intermediate product has been produced and added to the final result for each Quad in the first number. Figure 3 demonstrates the multiplication of 1.2345678E5 and 9.8765432E10.

```
                  Quads       Exponent
          x   1.234 5678 0000     +5
          y   9.876 5432 0000     +10

       1st Quad Product            2nd Quad Product
     1.234 5678 0000  +5         1.234 5678 0000  +9
  x       5432.0000  +3       x       9876.0000  +7
  ----------------------      ---------------------
    6.706 1722 8960  +11        1.219 2591 5928  +16

          Final Product = Sum of Quad Products
           x*y   1.219 3262 2100 2896  +16
```

Figure 3: Multiplying Exnumbers 1.2345678E5 and 9.8765432E10

The division algorithm should also be familiar to everyone. First, the result's exponent is calculated by subtracting the divisor's exponent from the dividend's exponent. Next, the divisor and dividend are normalized and they are forced to be positive numbers. This step is equivalent to lining up the divisor and dividend prior to beginning a manual division. Once again, two nested loops are used but now the outer loop iterates over all the digits in an ExNumber while the inner loop increments a quotient counter and subtracts the divisor from the dividend as long as the dividend is greater than or equal to the divisor. This is roughly what we do when manually comparing the divisor with the dividend and estimate a quotient which when multiplied by the divisor and subtracted from the dividend leaves the dividend less than the divisor. Our algorithm here, however, replaces the multiplication/subtraction step with just a series of subtractions. Finally, the divisor is divided by 10 to shift it within the range of the dividend for the next digit of the quotient and the whole process repeats. The division algorithm's outer loop guarantees that enough quotient digits are produced to fill all the ExNumber Quads. Figure 4 takes you through the steps involved in dividing 3.550E2 by 1.130E2.

```
     Quads    Exponent                Quotient Digit  Remainder
x   3.550      +2                           3          1.6  -1
y   1.130      +2                           1          4.7  -2
                                            4          1.8  -3
                                            1          6.7  -4
 Accumulated Quotient Digits                5          1.05 -4
x/y = 3.14159292035398                      9          3.3  -6
       230088495575221                      2          1.04 -6
       238938053097345                      .             .
```

```
1327433                                 .             .
                                        .             .
                                        4          3.8  -50
                                        3          4.1  -51
                                        3          7.1  -52
```

Figure 4: Dividing ExNumber 3.550E2 by 1.130E2

1.10 String Conversions

String Conversions

Before getting on to the actual calculator project, a couple of procedures are still missing. We need a way of translating a string into an ExNumber and, conversely, producing a string from an ExNumber.

The StrToExNum procedure accomplishes the first of our goals. This algorithm parses a string into a set of digits (0-9), signs (+, -), and punctuation (., E). First, leading spaces are stripped off and the sign of the number is determined. Then, as each digit is encountered, it is packed into a Quad at a position just to the right of the previous digit. The ExNumber's Quads are indexed by a digit counter which is also used to let us know when enough digits have been gathered. An exponent counter is incremented for each digit in the number. When reaching the digit maximum, only the exponent counter continues to be incremented but no more digits are added to the ExNumber. When a decimal is reached, the exponent counter increments are stopped. If an 'E' (exponent) is encountered, the ExNumber's mantissa is considered complete and the exponent's sign is determined. All following digits are merged into the ExNumber's exponent to which the exponent counter is either added or subtracted--depending on whether the exponent is positive or negative.

The second conversion routine, ExNumToStr, produces a character string from an ExNumber. Two different floating point number formats are supported: scientific notation (e.g., 1.23E10) and floating point notation (e.g., 234.234). Floating point notation is used whenever the ExNumber is small enough to be represented in a field of 'MaxDigits', which represents the maximum number of digits selected by the user. Both conversions begin by checking the sign of the ExNumber and inserting a minus sign if the number is negative.

The scientific notation conversion continues by rounding the ExNumber to the number of decimal places specified by the 'Decimal' argument. The leftmost digit is then inserted into the string, followed by a decimal point. Exactly 'Decimal' digits are placed after the decimal point (even if they are all zeros). The exponent symbol, 'E', is added to the output string, followed by the exponent's sign. A Modula-2 library function called ConvNumToStr, which converts integers into strings, is then used to convert the exponent into a string which is appended at the end of the output string.

Converting ExNumbers into floating point notation is a bit more complicated. As before, the number is rounded to maximum number of digits--in this case, the exponent size plus the number of specified decimal places. If the ExNumber is less than zero, a leading '0.' is placed in the string, followed by enough zeros to reduce the number's exponent to zero. Next, enough digits are placed

into the output string to satisfy the requested number of decimal places or exhaust the total number of digits in an ExNumber, whichever comes first. While placing these digits in the output string, a counter (InCnt) also keeps track of the decimal point so it can be placed at the right place in the output string. The last step of the conversion process removes trailing zeros from numbers like 35.123000000 to give more readable numbers like 35.123.

1.11 The CLI Calculator

The CLI Calculator

The Calculator module gives the complete calculator source code which supports the basic mathematical operations discussed here. The calculator is implemented with a very basic tokenizer (GetToken), which takes a stream of input characters and translates them into the set of tokens identified at the top of the Calculator module following the 'Tokens' type.

The stream of tokens drive a simple recursive expression evaluator (Expression) which accepts prioritized infix notation with bracketing limited only by stack size and input string length restrictions (250 characters). Operators are ordered as follows:
Highest priority (reciprocal, squared), Medium priority (times, divide), Lowest priority (plus, minus, negate, and unary plus). Thus, an expression like 2 + 5 * 6 would give the expected result of 32, and not 42.

The calculator also supports 'friendly' number entry which allows numbers to contain punctuation characters like commas, apostrophes, and underscores which can be used to separate groups of digits (e.g., 5,000 and 1'000'000.45). This feature is especially useful with 50-digit numbers!

1.12 A Sample Session

A Sample Session

To use the calculator, make sure you either are in the directory which contains the calculator program or copy the calculator into a directory which is part of your regular command path. Type 'Calculator' from the CLI and you are greeted with a 'CALC>' prompt. Simply type in the following equation exactly as it appears (hold down the ALT key and press '2' to get the ^{2} character and similarly to get the ^{1} hold the ALT key and press 'N', then '1'), and enter a Return:

```
4 * Pi * (89.234 + 4E1 / 2.0)$^2$ - (2-$^1$ * 56.78 * 10)
```

The answer 149658.8731 appears on the next line after the equation. Note the use of the name 'Pi' to represent the mathematical quantity 3.141592..., the squared operator (^{2}), and the reciprocal operator (^{1}). These extensions were trivial to add to the calculator and extend its usefulness. This example also shows a variety of number formats: integer, floating point, and scientific notation.

The default settings for the calculator give floating point number results.

If you want to fix the decimal point, just type 'DEC 2', for example, to set the number of decimal points to two digits. To restore to floating point format, type 'DEC 0'. To toggle between floating point and scientific notation type 'SCI'.

Experiment on your own with the calculator to see how it handles various errors like illegal characters and mismatched brackets. If you have an Oberon-2 compiler, attempt some extensions to the calculator, to recognize additional mathematical constants like the base of the natural logarithm (e) or attempt a cubed (x^3) operator.

1.13 ExIntegers

ExIntegers

The most obvious extension for the ExNumbers is a way of performing logical operations on them. Doing so requires that we restrict the vast dynamic range (i.e., -9.9E10000 to 9.9E10000) of the ExNumbers into a more manageable range that can be exactly represented within the 52 digits or so of ExNumbers. As well, to give easily discernable ExInteger limits when performing based arithmetic, a further constraint is imposed so that ExIntegers are within the range -(2**172) to 2**172 or -5.98E51 to 5.98E51. ExIntegers can thus exactly represent any 172-bit integer.

The simplest implementation of ExIntegers to support logical operations in Modula-2 is a mathematical set implementation. ExIntegers are built up using an array of the 16-bit set data type (SET).

1.14 ExInteger Conversions

ExInteger Conversions

To give us logical operations on ExNumbers several conversion routines are defined which translate ExNumbers into ExIntegers and vice versa. These conversions are hidden from the user of the ExInteger functions so that the parameters which are passed in and out of the procedures are always seen as ExNumbers. Thus, the ExInteger package interface is simplified so users don't have to perform explicit conversions every time they wish to perform a logical operation on ExNumbers. We see later that this calling convention simplifies the interface to the Calculator as well.

The ExNumbToExInt procedure near the end of the ExIntegers module, converts ExNumbers to ExIntegers. This algorithm first constrains the ExNumber to the valid ExInteger range (i.e., -(2**172) to 2**172). Next, a loop generates ExInteger set elements by taking the modulo 2**16 remainder of the ExNumber to effectively strip out a 16-bit chunk of the ExNumber and then type-casts this number into a 16-bit set (LONGBITSET), stored in the ExInteger. After each loop iteration, the ExNumber is divided by 2**16 and truncated to an integer to give access to the next 16-bit chunk. This loop terminates when all the ExNumber Quads are zero.

The inverse operation of converting ExIntegers to ExNumbers is performed by the

ExIntToExNumb procedure. A similar loop scans through the ExInteger chunks, in reverse order (i.e., from highest to lowest), converts each set into a 16-bit unsigned number, multiplies an accumulated total by 2**16, and adds the converted number to the total. The conversion is complete as soon as each ExInteger set has been addressed.

Based String Conversions

To enable the calculator to deal with numbers in other bases (e.g., hexadecimal, binary, and octal) I need to introduce two new procedures called StrToExInt and ExIntToStr. The first of these routines converts a based string into an ExNumber and the second routine performs the inverse operation of converting an ExNumber into a based string. Both procedures work only with integers: StrToExInt returns an illegal number error if requested to convert a floating point string and ExIntToStr constrains the ExNumber to a legal integer range before performing a conversion. I won't go into the details of these algorithms since they are very similar to the earlier string conversion routines you saw in the ExNumbers module. The chief difference is that the divisor becomes a power of the conversion base instead of a power of ten. The ExIntegers module has the complete source code for StrToExInt and ExIntToStr if you are interested in the algorithms used for these conversions.

1.15 Logical Operations

Logical Operations

The ExInteger module performs the standard logical operations of AND, OR, XOR, NOT or one's complement, bit setting, bit clearing, bit toggling, logical shifts, arithmetic shifts, and rotations. These functions are grouped according to the algorithm which implements each operation. For example, the AND, OR, XOR, and NOT functions all call the LOp procedure to perform the detailed logical processing of the ExInteger. For this reason, I just describe the central procedure for each grouping (i.e., LOp in this case) with the understanding that the other procedures which also use this algorithm have similar properties.

The ExAnd procedure serves as the representative of the first grouping which calls the LOp procedure, by passing in a customization function (the And function) which returns the intersection (equivalent to logical AND) of two BITSET arguments. The LOp procedure first translates the two operands to ExIntegers; then the passed function is used, on a 16-bit chunk basis, to logically AND (in this case) together both ExInteger arguments. The result is converted back to an ExNumber and is returned to the ExAnd procedure.

The second class of operations uses the LBit procedure to perform single-bit manipulations such as setting, clearing, and toggling bits. The ExSetBit procedure is used as an example to illustrate the general bit algorithm. As before, the ExNumber is converted to an ExInteger. LBit then makes use of the power procedure 'xtoi' from the ExMathLib0 module (described below) which implements the raising of the ExNumber, x, to the ith integral power, where i is an integer. The xtoi routine is used here to produce a single-bit mask based on the principle that 2**n sets the nth bit of an integer. In this case, the bit mask is ORed with the ExInteger, using the passed 'Oper' function in a call to LOp. Consequently, the ExNumber returned by this procedure, after conversion from an ExInteger, has its nth bit set.

Shifting operations are more awkward in an ExInteger format so they are implemented as multiplications and divisions by powers of two on ExNumbers. There are three different flavours of shifting algorithms: signed or arithmetic shifts, unsigned or logical shifts, and rotations.

The simplest shifting operation is the logical shift as implemented by the LShift algorithm. The ExNumber is first constrained to a valid ExInteger range. Next, if the bit shift quantity is greater than MaxBase2Bits (172), a zero is returned and the algorithm is aborted since the number has been shifted out of the ExInteger number range; otherwise, a shift mask is calculated using xtoi, and, for the ExShl procedure, is multiplied times the number to be shifted. This shifting operation is characterized by the equation Result = n * 2**b, where n represents the number to be shifted and b represents the number of bit positions to be shifted.

The rotation operations, implemented by the LRotate procedure, are slightly more complicated because the bit which is rotated out of the ExInteger range must be wrapped around and shifted back into the ExNumber. To help sense the state of a given bit in an ExNumber, the IsBitSet function was created. It forms a mask for a selected bit, ANDs this mask with a number, and then returns true if the bit was set. For ExROR, LRotate calls this function to extract the least significant bit before shifting the number. After the shift, if the detected bit was set, the most significant bit of the result is set using the ExSetBit procedure. The above process is repeated until as many bits have been rotated as were specified by the 'bits' parameter. Note: The worst case shift has been reduced, using a modulo operation, to the number of bits in an ExInteger since rotations always preserve the original number.

The final shifting procedure, ExAshr, performs an arithmetic shift right of an ExInteger. What this means is that the sign bit of the ExInteger is replicated each time the ExInteger is shifted right so that the number's sign is preserved. Since ExNumbers are implemented with a separate sign bit, this value is easily extracted by setting a SavedBit flag if the sign is negative. The ExInteger is then shifted right one bit at a time (using ExDiv by two) until 'numbits' have been shifted. For each shift, if the SavedBit flag was set, the upper bit of the ExInteger is set using ExSetBit to restore the number's sign.

1.16 ExNumber Math Library

ExNumber Math Library

Everyone knows that a calculator has transcendental (e.g., sin, cos, tan), logarithmic (e.g., log, ln), and power (e.g., x**y) functions. But these operations are usually very costly in terms of performance and have algorithms which can become very complicated--especially since our calculator has up to 52 digits of precision. In fact, during my literature search, the best algorithms I could find had only from 16 to 24 digits of accuracy. There were a number of alternatives: 1) come up with the algorithms from scratch which would give 52 digits of accuracy; 2) use a lower-accuracy algorithm; 3) use existing lower-accuracy functions from the Amiga's IEEE math libraries. I opted for the third choice since I didn't have the time to invest in producing and testing the required precision algorithms and the speed penalty could be horrendous. As well, there was no point in reinventing the wheel when algorithms of

comparable precision already existed on the Amiga.

I essentially created an interface (ExMathLib0 module) to the double-precision IEEE floating point library. The calculator could thus have 15 digits of precision at hardware speeds (if you have a floating point coprocessor). Several functions such as square root, cube root, integral powers/roots, and factorial do have the full ExNumber precision because the algorithms were easily extended to give 52-digit accuracy. If you have the ability and time to extend the precision of any other functions, I would appreciate hearing from you so that I can update this module with the new algorithms. Any algorithms I receive will be placed in the public domain--with the author's permission.

The heart of the IEEE floating point interface lies in the the ExNumToLongReal and LongRealToExNum conversion routines. To simplify the conversion process and demonstrate the power of reuse, I used several compiler-supplied conversion routines, ConvStrToLongReal which translates a string into a double-precision IEEE floating point number; and ConvLongRealToStr which performs the inverse operation. As well, ExNumToStr and StrToExNum, from the ExNumber module, provide string to ExNumber translations.

An IEEE number is converted to an ExNumber through an intermediate step of translating the number into a string. StrToExNum takes this string and produces a valid ExNumber. To reverse this process and produce an IEEE representation from an ExNumber, ExNumToStr uses an ExNumber to produce a string which ConvStrToLongReal then translates into the double-precision IEEE floating point number. The expX procedure shown below demonstrates the conversion process and the IEEE interface. The expD function is a compiler-supplied library function which ties directly into the Amiga's double-precision IEEE library.

```
PROCEDURE expX(VAR Result : ExNumType; x : ExNumType);
BEGIN
  LongRealToExNum(expD(ExNumToLongReal(x)));
END expX;
```

While many routines can be obtained using the IEEE library, some, like the inverse hyperbolic trigonometric operations, are not available in this library. I had to develop these algorithms from their basic definitions which follow:

```
ArcSinh(x) = Ln(x + Sqrt(x*x + 1))
ArcCosh(x) = Ln(x + Sqrt(x*x - 1))
ArcTanh(x) = Ln((1 + x) / (1 - x)) / 2
```

where Ln represents the natural logarithm of a number and Sqrt represents the square root of a number.

Several other functions such as integral roots and powers have algorithms which were easily extended to give full 52-digit precision. The integral root algorithms are based on Newton's iterative method of finding roots of a function whose basis equation is y(n+1) = y(n) - f(y)/f'(y) where y(n+1) is the (n+1)st iterative solution, y(n) is the nth solution, f(y) is the function whose root is required, and f'(y) is the derivative of f(y). I applied this equation to obtain the general root-finding algorithm shown below:

```
y(n+1) = (y(n) * (r - 1) + x / y(n)**(r - 1)) / r
```

where y(n+1) and y(n) are defined as before, r represents the root power (e.g., r = 2 for a square root), and x is the number whose root we wish to determine. The Root procedure (top of ExMathLib0 module) implements this general algorithm and also adds the capability of finding negative roots (e.g., the cube root of -8 is -2). Both the sqrtX and rootX exported procedures use this general-purpose Root routine.

Integral powers are calculated using an algorithm published by Donald Knuth in his work, "The Art Of Computer Programming", the second volume. I have adapted his algorithm to also work for negative powers. The resultant implementation is shown in the xtoi procedure in the ExmathLib0 module. The beauty of Knuth's approach is that the calculation of any integral power involves only about log(n)/log(2) multiplications where n represents the number's integral power. For example, this algorithm calculates 15**64 using only six multiplications! The expX and powerX procedures use the xtoi routine whenever they evaluate integral powers.

The last routine for which I have an extended precision algorithm is the factorialX procedure which computes the factorial of a number. Because ExNumbers have a much larger dynamic range (-1x10**10000 to 1x10**10000) than most other floating point numbers, factorials can be calculated of numbers as large as 3249! Compare this with the typical calculator which only gives factorials as large as 69! However, calculating this large a factorial also normally requires 3248 multiplications which could take quite a while even on the fastest Amiga. To reduce this time, precalculated factorials of 500!, 1000!, 2000!, and 3000! have been stored in the ExMathLib0 module. Thus, to calculate 3249!, only 249 multiplications are required since the algorithm starts with 3000!. Calling a routine 1000 times recursively can take a lot of stack space so the factorial procedure calculates factorials using an iterative algorithm rather than the recursive algorithm everyone is taught in school to keep the calculator's stack requirements to the CLI default.

1.17 Calculator Features

Calculator Features

Remembering

The first addition is that the calculator now can store and recall up to sixteen ExNumbers using the syntax: x STM n where x represents a number or expression to be stored and n is the location (0 to 15) where the result should be stored by the StoreMemory routine. To recall the number type Mn where n represents the ExNumber location to recall via the RecallMemory routine. The ExNumbers are stored in a simple array which can be easily extended to allow number storage which is only limited by available memory.

All these storage locations and other calculator state variables are stored to the RAM: drive (via the StoreState procedure) between calculator invocations so results from previous calculations can be reused during later sessions. The persistent memory also helps get around the problem of having expressions which are longer than the maximum allowable input string of 250 characters. They can simply be split up and calculated in pieces, with intermediate results stored in the calculator's memory.

Argument Interface

In addition to the interactive mode, it is possible to use the calculator much like the Amiga's Eval program where the expression to be evaluated is passed to the calculator when it is invoked. For example, typing 'Calculator 2^10' produces the result 1024. The command line argument is extracted by the GetCLI procedure and then is processed just as if you had typed the expression interactively. Because the memory is retained between calculator invocations, you could store the results of one calculation in memory and then use those results in a following calculation. Remember that command line arguments are automatically separated by the operating system if spaces are left between words so quotation marks must be placed around expressions. For example, to calculate the sum of the first five factorials, from the CLI, type:

```
Calculator "1! + 2! + 3! + 4! + 5!"
```

The answer '153.' is displayed when the CLI prompt returns. Of course, the spaces are optional, and 1!+2!+3!+4!+5! (with no spaces) would produce the same result without requiring quotations.

Output Formats

You can toggle the format of the calculator's output numbers between the default floating point notation and scientific notation. Just type SCI to toggle between these two modes. Be careful to type the exact name as shown because all the calculator functions are case-sensitive. If you type in a number like 2 and then switch to scientific notation you may be shocked at all the trailing zeros that get displayed. Several commands let you suppress these extra digits: DP n lets you enter the number of decimal point digits which should be displayed where n can be a number between 0 and 52. Specifying a value of 0 selects the default floating decimal point notation while any other number fixes the decimal point at n digits. DIG n selects the number of digits that the calculator uses when performing its calculations. Valid values for n are from 0 to the default of 52. All calculator format definitions are saved to the RAM: disk between calculator sessions.

Other Functions

The calculator allows evaluation of any trigonometric function (SIN, COS, TAN). The default angular units are in degrees. The command DRG toggles between angular units in degrees, radians, and grads.

Based numbers, as discussed above, represent a subset of the ExNumbers. To get the calculator into the based number mode, type BAS n where n represents the numerical base from 2 to 16. The value for n is always specified in decimal notation no matter which base the calculator is using. Numbers containing decimals and exponents are illegal when in a numeric base other than 10. Based numerical systems greater than 10 use the uppercase alphabetic characters A-F to represent numbers in addition to the standard digits but every number must always begin with a valid base digit from 0 to 9; numbers beginning with the digits A-F require a leading 0. Underscores, apostrophes, and commas may be used to separate groups of digits no matter which based representation is being used.

Table 2 below summarizes the calculator operations and commands along with the required syntax when accessing the calculator from the CLI. An additional restriction is that the longest allowable expression string cannot exceed 250 characters.

+	Addition
-	Subtraction
*,$\times$	Multiplication
/,$\div$	Division
^{2}	Squared
^{3}	Cubed
-^{1}	Reciprocal
()	Brackets
^, **	Power
%	$\times$ 0.01
&, AND	Logical And
|, OR	Logical Inclusive Or
XOR	Logical Exclusive Or
CPL	Logical Complement
MOD	Modulo
DIV	Integer Division
SQRT	Square Root
CBRT	Cube Root
ROOT	Any Root
e	Natural Log Base
e^	Power of e
LN	Natural Logarithm
LOG	Base 10 Logarithm
(A)SIN	(Arc)Sine
(A)COS	(Arc)Cosine
(A)TAN	(Arc)Tangent
(A)SINH	(Arc)Hyperbolic Sine
(A)COSH	(Arc)Hyperbolic Cosine
(A)TANH	(Arc)Hyperbolic Tangent
SBIT	Set Bit
CBIT	Clear Bit
TBIT	Toggle Bit
SHR	Shift Right
SHL	Shift Left
ASR	Arithmetic Shift Right
ROR	Rotate Right
ROL	Rotate Left
Mn	Memory Location n
STM n	Store to Memory n
Pi	Constant Pi
SCI	Toggle Scientific/Floating Point
BAS n	Change to Base n
DIG n	Use n Digits
DP n	Use n Decimal Places
DRG	Toggle Degree/Radian/Grad

Table 2: Calculator Operations and Commands

1.18 Graphical Interface

Graphical Interface

Probably the most daunting activity when designing software has to be the creation of the graphical interface. User interfaces typically make up about 60 percent of the entire application's code and are very labor-intensive since they require many iterations of compile-link-execute cycles to get just the right placement of 'gadgets' (i.e., buttons and text fields). Fortunately, a program called CanDo (see back issues of Amazing/Amiga Computing) provides a simpler alternative to the otherwise painful process of defining a program's GUI.

I designed the basic arrangement of calculator keys to give a central numeric cluster with less used keys positioned to the sides of the numeric cluster. This calculator arrangement is typical of most commercial calculators with the exception that I opted for a horizontally extended keypad instead of the more common vertically extended keypad to maximize the use of the available screen space and give more space to the equation display, located just above the keypad. To reduce typing (or clicking), a scrollable region is located just above the display area which shows all the previously entered equations. You click on one of these equations to bring it back into the equation display for editing or recalculation.

CanDo V2.01 provides a new tool called 'SuperDuper' which can duplicate a single gadget like a button or text field any number of times with both vertical and horizontal offsets. Thus, the above matrix of calculator keys is extremely simple to lay out. First, I defined a single button with dimensions of 50 by 15 pixels so that 54 of these buttons in a nine by six matrix would fill the window area. Using the SuperDuper tool with an x offset of 50 and y offset of 15, I duplicated this button to totally fill the window both vertically and horizontally. Buttons located in the central area were then deleted to make room for the central button cluster. Starting with a single button which was half the horizontal size of the function keys, I duplicated this button to create the six by six central numeric cluster.

The calculator key labels are composed of text strings written to the display using a custom font after the initial calculator window has been created. The Helvetica-like font contains special characters to allow display of radicals, exponents, and powers on the calculator keys. Unfortunately, CanDo doesn't support any font other than Topaz for use in a text field, so the equation field can't reproduce the key labels exactly. I was forced to substitute alphabetic abbreviations for the roots and powers. The equation recall list uses the same abbreviations as the equation field.

The result window/equation entry field consists of a left-justified text field outlined with a beveled border. Equations can be entered directly into this field via the keyboard or by a series of mouse clicks on the calculator keys.

Above the equation entry field is the equation/memory/result list which consists of a CanDo document tied to a scrollable list object. I update this list with either the most recently entered equation or a list of the current calculator contents or a history of the calculated results. The choice of what gets displayed is selectable via the History/Show menu items (more about the menus below). Only the displayed selection gets logged to the list

object by calculations so when displaying the equations, for example, the previous calculated results are not retained. Of course, memory locations are kept current and will always display the exact calculator memory contents.

To the left of the history list is the calculator status display which shows the current angular measurement system, the type of floating point display (either floating decimal point or scientific notation), the number of decimal places, the numerical base (from 2 to 16), and the number of significant calculator digits in use. The five buttons immediately below this status display control these calculator attributes.

Three menu bar items: Project, Custom, and History give access to project items such as iconification, printing, and an 'about' box; item list display, clearing, and printing; and calculator customization options including selection of the processor-specific calculator (i.e., 68000, 68020, or 68030). The menu-based customization options duplicate the left-most calculator keys to a degree, although they are less flexible.

When I defined the calculator key labels in the window startup script, I also added a string to the key which gets appended to the active equation string via a call to "ProcessKey" when you click on a calculator key. This same string also gets displayed in the equation entry field. CanDo doesn't allow the setting of the cursor in the text field so, unfortunately, the cursor doesn't follow along when new text is appended via key clicks. When entering equations from the keyboard, everything works as you would expect.

There are some complications to this approach. For instance, when you have entered a <Return> or clicked on the Equals key, you expect the following key clicks to start a new equation and not just append to the end of the equation which was just calculated. The "AddKeyToDisplay" routine encapsulates these intricacies as follows:

```
*************
* Global routine "AddKeyToDisplay"
  If First
    Let First = False
    SetText "Display",Arg1
  ElseIf (ArgCount > 1) And LastKeyNumber
    SetText "Display",Arg1||TextFrom("Display")
    Let LastKeyNumber = False
  Else
    SetText "Display",TextFrom("Display")||Arg1
  EndIf
  If FindChars("01234567890ABCDEF.",Arg1,1) = 0
    Let LastKeyNumber = False
  EndIf
  SetObjectState "Display",On
* End of routine "AddKeyToDisplay"
*************
```

Here the "First" flag is set by other routines whenever the display should be cleared and a fresh equation is started (e.g., when the '=' key has been entered). The "LastKeyNumber" flag is used to prepend new strings to an existing equation in the case where a number was previously entered and then a function like 'SIN' is selected via a button click. For example, if you just entered the number '45' and then clicked on the 'SIN' key, the calculator equation would be 'SIN 45' instead of '45 SIN'. This simple

addition makes the calculator much more user-friendly since it automatically corrects a common mistake you might make.

1.19 GUI/CLI Interaction

GUI/CLI Interaction

The heart of the calculator interface is the "Calculate" routine which both submits the final equation string for computation to the external calculator and parses the entered equation in order to update the status display area.

As a first step, the equation is extracted from the "Display" text entry object and then passed to the "CallExCalc" routine to be processed as follows:

```
*************
* Global routine "CallExCalc"
  SetPointer Dir || "Brush/BusyCursor",5,5
  Dos State.Calculator||" >RAM:Result.txt " || Char(34) || Arg1 || Char(34)
  OpenFile "RAM:Result.txt","ResultBuffer",READONLY ,OLDFILE
  FileReadLine "ResultBuffer",Arg2
  Close "ResultBuffer"
  SetPointer
* End of routine "CallExCalc"
*************
```

In this routine I am invoking the calculator defined in the "State.Calculator" field (i.e., either a 68000-, 68020-, or 68030-based calculator) and redirecting the calculator's normal output to a RAM-based text file called "Result.txt". I then open this file from within CanDo, read the result, and return the calculated result to the caller of this routine via the second argument 'Arg2'. A busy pointer is also displayed as long as this routine is waiting for the external calculator to return an answer.

Note that CanDo's 'Dos' calling routine will wait until the calculator has calculated and placed the result in the output file and terminated its execution. I thus avoid synchronization problems in attempting to access the 'Result.txt' file which might otherwise occur if the CanDo script used the usual 'Dos Run' command sequence which does not wait.

To speed up the external calculator, I make the chosen calculator resident during CanDo's 'After Attachment' script with the following line:

```
Dos "Run >Nil: <Nil: c:Resident " || Dir || State.Calculator
```

As a second step, the 'Calculate' routine updates the history display and parses the equation string to determine whether the status area needs to be updated with a new decimal point, numeric base, or the number of digits. This step is essential to guarantee that the CanDo status area always accurately reflects the state of the external calculator.

If an error has occurred (i.e., the key word "Illegal" is contained in the external calculator's result), no status or history update is allowed to prevent erroneous history and status values.

1.20 CanDo Tricks

CanDo Tricks

Printing From CanDo

One disappointment with CanDo is its lack of a built-in print command. Since I wanted to be able to print the history list contents, I defined the following "Print" routine:

```
*************
* Global routine "Print"
  OpenFile "RAM:Print.dat","PrintBuf",WRITEONLY ,NEWFILE
  FileWriteChars "PrintBuf",Arg1
  Close "PrintBuf"
  Dos "Copy RAM:Print.dat PRT:"; Print the above file & wait
* End of routine "Print"
*************
```

The argument to this routine (‘Arg’) is written to an external text file and then this file is copied to the printer port via a ‘Dos’ copy. I had to go this indirect route because I wanted the CanDo script to wait until the given file had been sent to the printer because the print routine is called multiple times when printing all the history information. I thus prevented the possibility that a subsequent print command would attempt to open the ‘PRT:’ device a second and even a third time while it was still busy. This bug actually occurred during development and I was scratching my head for quite a while until I finally discovered the root of this problem.

Menu Tricks

The CanDo menu system poses some additional challenges, especially when attempting to give programs the "look and feel" of the V2.0 operating system. I was forced to simulate the separating lines, typically seen dividing unlike menu items, by using a text string consisting of a series of hyphens which I then disabled so that the menu item takes on the familiar ghosted appearance and is inactive when you attempt to select it.

Another limitation is that mutually-exclusive menu items (in which a series of alternatives can each be checked but two or more are not allowed to be active simultaneously) are not explicitly supported by CanDo. To implement mutually-exclusive menu items, I defined a routine in CanDo like the following:

```
  *************
  * Global routine "SetDigitsMenu"
    SetObjectState "8",Off
    SetObjectState "12",Off
    SetObjectState "16",Off
    SetObjectState "20",Off
    SetObjectState "32",Off
    SetObjectState "40",Off
    SetObjectState "52",Off
    If Match(Arg1,"8","12","16","20","32","40","52") > 0
      SetObjectState Arg1,On
    EndIf
```

```
  * End of routine "SetDigitsMenu"
  *************
```

which set the state of all the listed menu buttons to an 'off' condition. For checked menu items this means that the checkmark is removed. The caller of this clearing routine then toggles an internal flag and sets its own button checkmark to either 'on' or 'off' depending on the internal flag. Using CanDo's built-in toggle buttons would have been easier but I couldn't always guarantee that the checkmark imagery would be synchronized with the actual toggle state of the button. Keeping an internal toggle state flag avoids any problems.

1.21 GUI ExCalc Demo

GUI ExCalc Demo

Once the calculator is installed and displaying the GUI interface, you can use the mouse to click on any calculator keys to enter the corresponding symbol in the display object. Alternatively, you can use the Amiga's keyboard to enter equations by clicking on the display object and then typing normally from the keyboard. You can switch back and forth between using mouse clicks and keyboard entry, keeping in mind that the calculator is case-sensitive and expects function names in uppercase characters. When in doubt, click on the desired function to get the proper capitalization for that operation.

If the equation history log has been enabled via the History/Display menu, all the equations which you enter are displayed in the history area. Other History/Display menu options show the calculation results or the current calculator memory contents. Select the equation option to keep track of the equation you'll enter. Click on the 'CE' button and enter the equation:

```
SQRT(3^2 + 4^2)
```

which computes the length of the hypotenuse of a right-angle triangle with sides of length 3 and 4. Click on the '=' or enter a <Return> and the answer '5.' should be displayed. Notice that this equation has also become the first history element in the history display. Click on the 'CE' button again and click on the equation in the history display. It should reappear in the equation display. Click on the display object around the '3^2' and use the keyboard cursor keys to position the cursor over the '3'. Replace the '3' with a '5' and similarly replace the '4' with a '12' to get the equation:

```
SQRT(4^2 + 12^2)
```

Click on the '=' key to get the new result of '13.'. Now click on the 'CH' and 'CE' buttons to clear the history display and clear the answer.

I will be using memory location 2 (M2) to store the number whose root is being found. Initialize this location with the number 100 as follows:

```
100 STM 2
```

Similarly initialize memory location 1 (M1) with an initial guess of the root, in this case the original number divided by 2:

```
    M2 / 2 STM 1
```

Next enter the following equation which iteratively computes the square root of a number followed by an '=' (see the second article for details):

```
    0.5 * (M1 + M2 / M1) STM 1
```

where M1 is the current estimate of the square root and M2 is the number whose root is being found. The first iteration should display the answer '26.'. Recall the equation from the history display and click on '=' again. The second iteration gives '14.923....'. Keep recalling the equation and clicking on '=' until the answer '10.' is displayed after about four more iterations. From the History/Display menu options select the display of the calculator memory contents. Memory locations 0 to 15 are displayed in the history display with M1 set to '10.' and M2 set to '100.'. Clicking on a memory location in the history window recalls the contents of that location.

For the last example, make sure that the answer '10.' is still displayed and click on the 'BAS' key. Now click on '1' then '6' to enter 'BAS 16'. Click on '='. The status display shows that the calculator is now operating with base 16 numbers and the previous result of '10.' is now displayed as the converted number '000A'. Click on 'CE' and enter the following equation:

```
    (0'FFFF'0000'FFFF OR 1234'0000) AND 0'AAAA
```

from the keyboard and end with a <Return>. The answer '0FFFF1234AAAA' is displayed. Note the use of apostrophes to visually separate groups of four hexadecimal digits. This capability is extremely useful when entering very large numbers. The calculator also allows the use of a comma as a separator.

The memory contents have not been altered when the calculator switched bases. The reason for this is that fractional results could get truncated when working with numerical bases other than ten, so only numbers which are actually recalled from memory and used in a calculation are truncated.

1.22 Summary

Summary

I have just touched on a few of the calculator's capabilities in this brief demonstration. To fully appreciate all the calculator features, experiment on your own with the calculator and attempt some useful modifications of the supplied Oberon-2 source code or the CanDo decks which make up the user interface. Some suggestions are to give the CanDo interface the capability of storing in the history log both formulae and calculated results. The calculator could be extended with statistical functions and imaginary numbers. I would be interested in hearing about the uses you find for this calculator and any additions you make. Happy computing!

1.23 ExCalc.guide/M1_REGIS

Registration

As you may have noticed, ExCalc is a shareware product. All functions are available for testing without paying any money. However, the program will shut itself down after half of hour and will not restart until the next time you power on your computer. As well, shareware reminders will pop up while the calculator starts up and when it automatically exits. If you continue using ExCalc beyond an initial evaluation period of 30 days, you are required to mail your registration form along with $20 US.

When you register, you will receive an updated and personalized ExCalc program (without the shareware reminder screens and automatic power-down), the complete CanDo GUI source code, and calculator engines for both 68020 and 68030 processors.

To get registered for ExCalc, please print the file OrderForm on your printer (if no printer is available, please copy the information), fill out this form and send it to the given address. I will endeavour to act upon your registration within two weeks after I get the registration form and the shareware fee. In most cases it will be done faster. Your updated program will be shipped by (snail)mail.

Please contact the author directly for site licenses and other special licensing agreements.

The author reserves the right to refuse registration requests.

1.24 ExCalc.guide/M1_UPDAT

Updates

When you become a registered user, you will get the latest personalized release of ExCalc and will be entitled to receive one free update. Each update after that will cost $10 US for registered ExCalc owners to cover shipping, duplication, and material costs. Of course none of this applies if you have not registered.

1.25 ExCalc.guide/M1_RIGHT

Legal Stuff

Liability

Distribution

Trademarks

Copyright

1.26 ExCalc.guide/Liability

Libability

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDER AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

1.27 ExCalc.guide/Distribution

Distribution

Distribution of the shareware version of ExCalc and associated files are allowed on any data medium and can be made available on bulletin boards or other networks only if the original, unchanged, compressed file is distributed or the entire directory structure below is kept intact:

```
Fonts (dir)
     CalcFont1 (dir)
       13                              13b
     CalcFont2 (dir)
       11
     CalcFont3 (dir)
       8
     CalcFont4 (dir)
       11
  CalcFont1.font                  CalcFont2.font
  CalcFont3.font                  CalcFont4.font
Brush (dir)
  BusyCursor
Source (dir)
  Calculator.mod                  Calculator.mod.info
  ExIntegers.mod                  ExIntegers.mod.info
  ExMathLib0.mod                  ExMathLib0.mod.info
  ExNumbers.mod                   ExNumbers.mod.info
  InOutExt.mod                    InOutExt.mod.info
  TestExNum.mod                   TestExNum.mod.info
```

```
  Brush.info                    Calculator
  Calculator.info               Disk.info
  ExCalc                        ExCalc.guide
  ExCalc.guide.info             ExCalc.info
  Fonts.info                    OrderForm
  OrderForm.info                Product-Info
  Product-Info.info             Source.info
```

It is also allowable to levy copy charges for the distribution on floppy disks or CD-ROMs, as long as it has been stated clearly for the user that (s)he has not thereby paid for the shareware fee. It is not permissible to copy, distribute or reproduce the non-shareware versions of ExCalc and/or the calculor engines without obtaining the written permission of the author.

Special Versions

If you want me to program a special version of ExCalc e.g. for commercial release as part of any commercial software package, please contact the author directly.

1.28 ExCalc.guide/Trademarks

Trademarks

ExCalc is a trademark of Computer Inspirations.

CanDo is a trademark of Inovatronics.

1.29 ExCalc.guide/Copyright

Copyright

1.30 ExCalc.guide/M1_BUGRP

Known Bugs

There are no known bugs. If, however, you find a problem, please forward to the author a complete description of your machine configuration along with details on how to reproduce the bug. If the problem can be duplicated, a free upgrade patch of the ExCalc software or a description of a work-around will be shipped to the person who first reports the problem. Note: This offer only applies to registered ExCalc owners.

The author has no obligation to fix alleged bugs that cannot be duplicated.

1.31 ExCalc.guide/M1_FUTUR

Future

The following list gives an idea of some features future releases of ExCalc may have. The ordering of this list is not significant.

- Add graphical plotting of equations
- Automatic logging of both formulae and calculated results.
- Metric conversion support.
- GUI integrated with the calculator.
- AREXX interface to the calculator engine.
- Speed improvements for extremely large numbers.
- Improved accuracy algorithms for transcendental functions.
- Statistical function support.

Please contact the author if there are some features which you consider more important than others. As well, suggestions for additional features are welcome, but no guarantees regarding implementation can be made.

1.32 Author

Author

Michael Griebling
c/o Computer Inspirations
150 Clark Blvd., Suite One
Brampton, Ontario
Canada, L6T 4Y8

Tel. (416) 840-4648

e-mail: grieblm@trt.allied.com
or mgriebling@bix.com