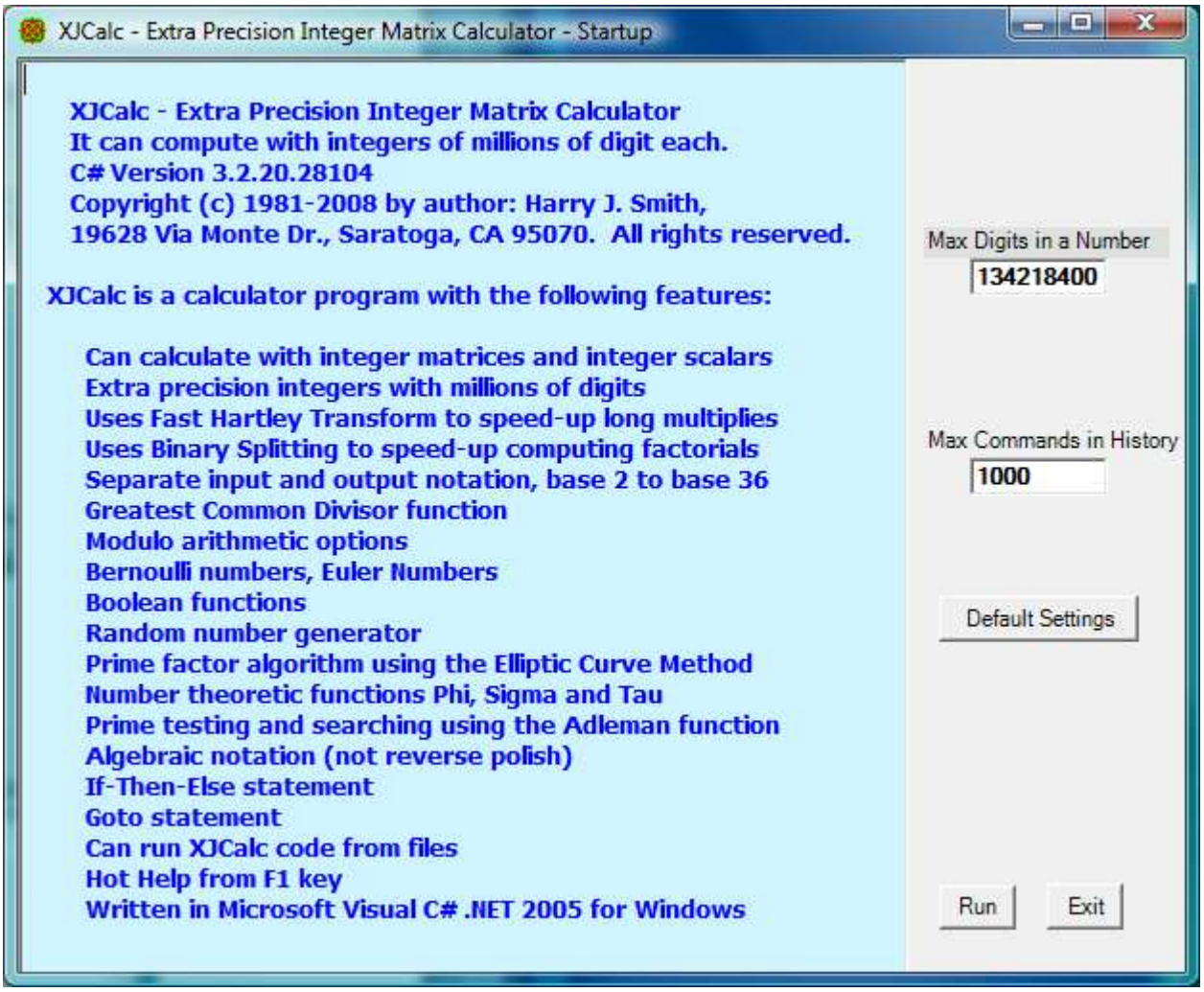
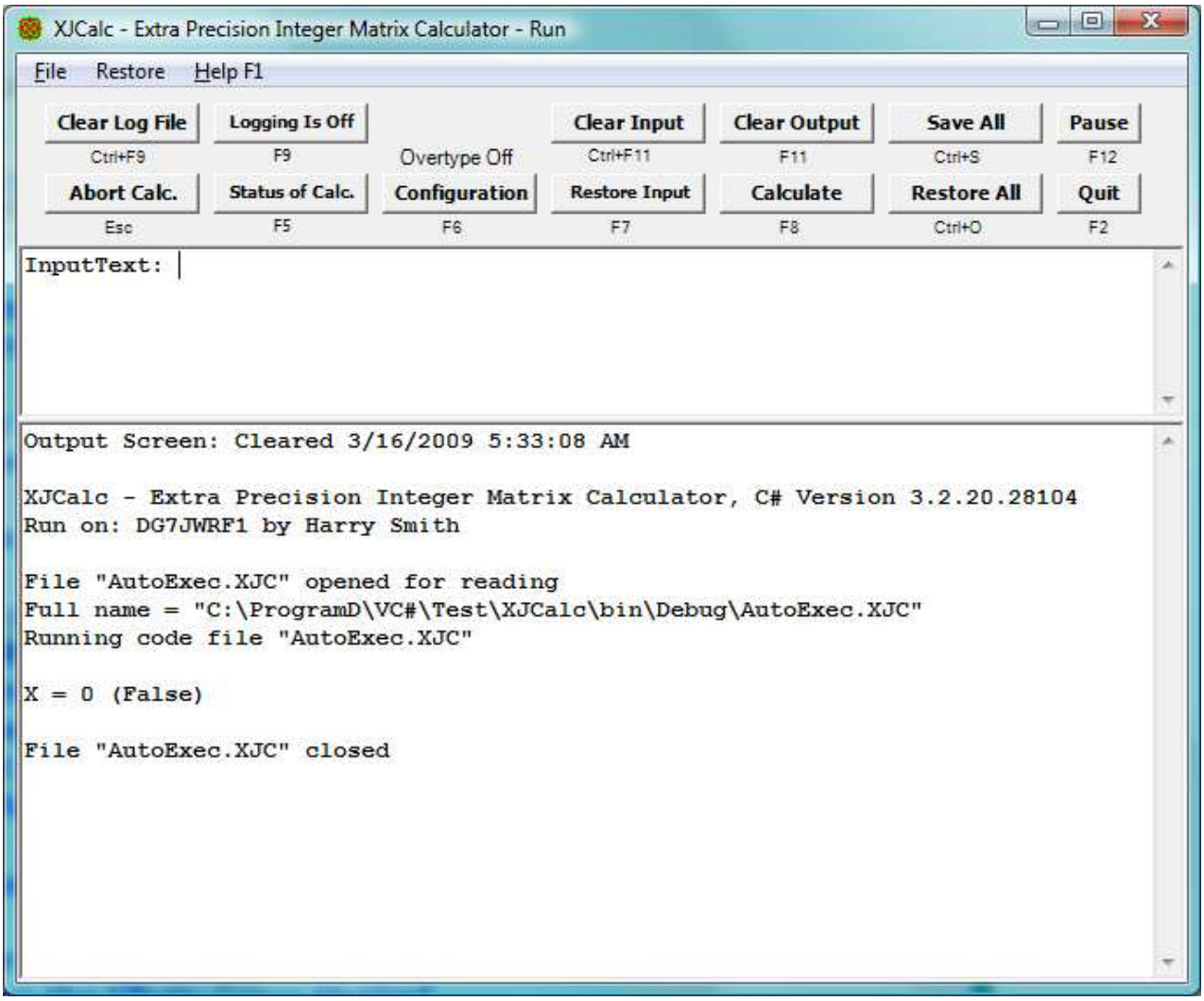
This document applies to the C# Version **3.2.20.28104** of XJCalc,

Introduction -

When you execute the program XJCalc.exe it responds by displaying the Startup form:



Just click the Run button and the Run form is displayed:



Numbers are stored in memory in decimal; actually they are stored in base 100,000,000 as an array of super-digits. Each super-digit is stored as a 32-bit integer between 0 and 99,999,999. As variables change in value, memory is dynamically reallocated so no more memory is used than is needed to represent their current precision.

At any given time there is a current max decimal digits that will be allowed for an integer and a max decimal digits to ever allow in an integer. These values are initialized to 134218400 and 134218400 respectively and can be changed after the program is running. The current max can be changed by the M primitive or the SetMax procedure. The max ever allowed can only be changed by changing the Max Digits in a Number field on the Startup form.

The Run form has two large text boxes. The upper one is for command input. Commands may be entered one at a time each followed by a return, or several separated by one or more blank spaces or semicolons. A separator is never needed between primitive op codes and is only needed otherwise to prevent ambiguity of meaning. Commands are not case sensitive, upper and lower case letters are always interpreted the same.

There are four basic types of commands: 1) Enter a number, 2) Execute a primitive op code, 3) Evaluate an equation, and 4) Do a procedure.

The calculator contains a list of named numbers or variables. Initially the list contains only the item X = 0. Its name is X and its value is 0. Items can be added to the list by evaluating an equation. Equations are assignment statements like {variable} = {expression}. A {variable} is a name of a variable and an

{expression} is an expression of terms, factors, functions, variables, constants, and {expression}s. Item names are alphanumeric with the first character alphabetic, have all characters significant but not case sensitive.

A given named numbers on the list can be a scalar or a matrix. A scalar is an integer. A matrix is a rectangular array of scalars arranged in r rows and c columns. The element in row i and column j is labeled as element i, j with 1 <= i <= r, 1 <= j <= c. A one by one matrix is not considered to be a scalar and a scalar is not a matrix. An m by 1 matrix is a column vector. A 1 by n matrix is a row vector.

Parentheses can be nested to any level in expressions. Any number of closing parentheses can be replaced with a single semi-colon or an end-of-line. Thus x = (a / (b * (c + d; is a legal assignment statement and is interpreted as (a / (b * (c + d))).

Any time a variable is referenced that is not currently on the list, it is added to the list with a value of 0.

At any given time, one item on the list is the active item. This is referred as Top, the item on top of the list. Initially item X is the active item. When an expression is evaluated, the variable being assigned a value becomes the active item. If the {expression} part of an assignment statement is left blank, like X=, the referenced variable becomes the active item without changing its value.

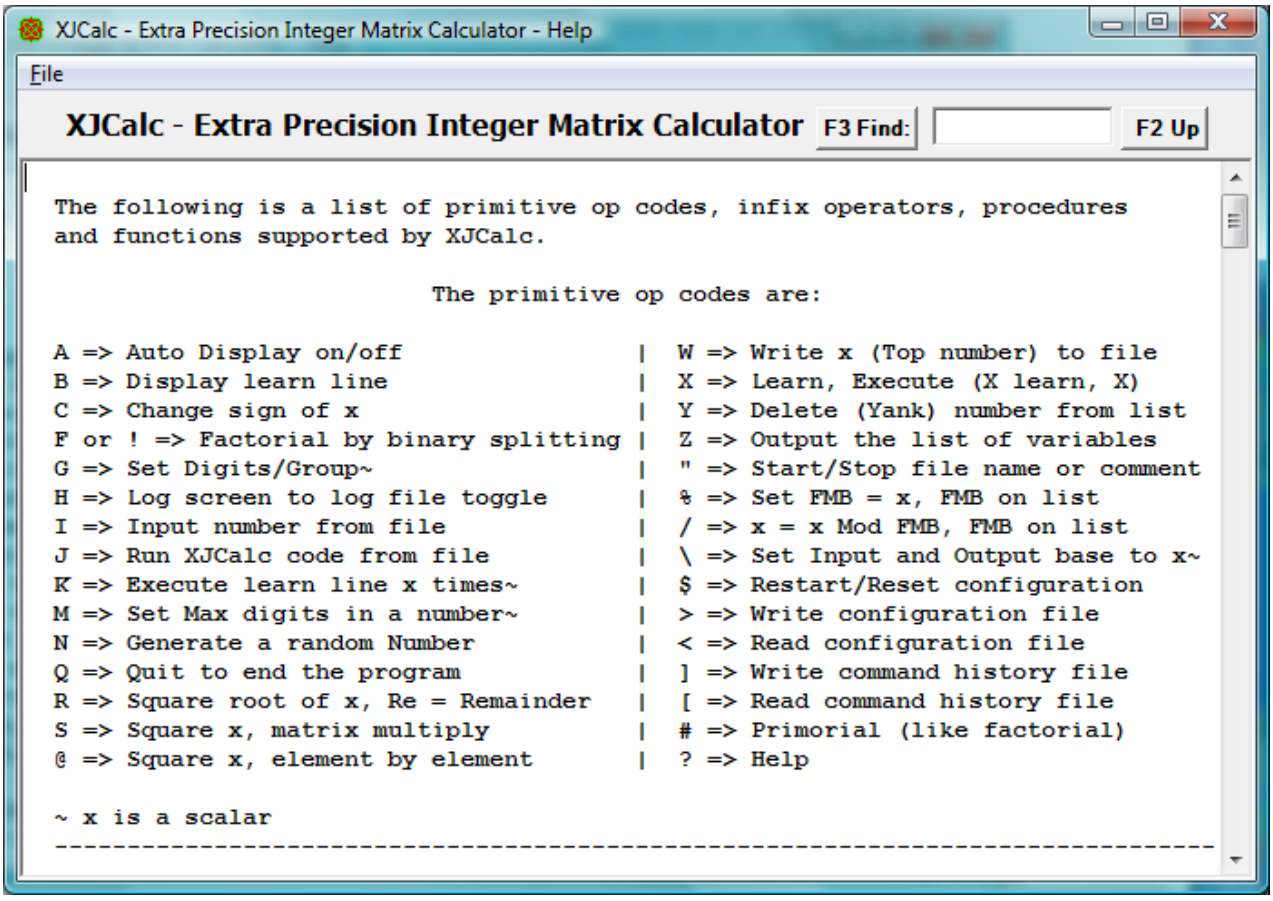
When a number is entered by itself, it replaces the value of the active item. Numbers (constants) may have a leading sign and embedded commas. An example of a constant is -12,345678,9. The commas and plus signs are optional. Because commas are allowed in input constants to make them readable, a single comma cannot be used to separate numeric arguments in functions calls. An example of this is: X = Mod(12,345'6,789). A tic mark separates the two arguments instead of a comma as is normally done. A ", " also works as in Mod(12,345, 6,789).

There is a special feature that allows for the entry of an equation without the "X=" preceding it. If the first character of a command is ">= '0' and <= '9' and <= MaxDigit" or equal to "(", "-", "+", or "=", the command will be prepended with "Top.Nm=" (no == though), where Top.Nm is the name of the item on top of the number list. This is done for each command of a multi-command command line. This also works for the other assignment operators (+=. -=, *=, /=, and %=). For example, +=Y will be treated as X = X + (Y). Y is evaluated and added to the top item on the list.

If the program is executed from the DOS prompt with one or more arguments, the initial Startup form is not displayed and the arguments are taken as an initial XJCalc command line. This allows you to control the execution of XJCalc from batch files and XJCalc code files with no operator intervention. The XJCalc code file AutoExec.XJC is always run first, even before the DOS command line commands. The ASCII Tab character (9) is allowed in XJCalc code files and in the initial command arguments and treated as a blank space.

Special handling is given to the first argument on the DOS command line. If it ends in .XJC, it is changed to Run("... .XJC") so this XJCalc code file will be run. If it ends in .XJN, it is changed to ReadN("... .XJN") so this XJCalc number file will be read and added to the list of items. This allows XJCalc to be run by Windows or a program, like ZTree, by associating the file XJCalc.Exe with the extensions XJC and XJN, and then opening a file with one of these extensions.

? => Help: the “?” primitive causes the Help form to be displayed:



The Help form is scrollable and resizable. It contains a list of primitive op codes, infix operators, procedures, functions supported, Function key actions, and commands used in XJCalc code files:

The primitive op codes are:

```
A => Auto Display on/off                | W => Write x (Top number) to file
B => Display learn line                 | X => Learn, Execute (X learn, X)
C => Change sign of x                   | Y => Delete (Yank) number from list
F or ! => Factorial by binary splitting | Z => Output the list of variables
G => Set Digits/Group~                  | " => Start/Stop file name or comment
H => Log screen to log file toggle      | % => Set FMB = x, FMB on list
I => Input number from file             | / => x = x Mod FMB, FMB on list
J => Run XJCalc code from file          | \ => Set Input and Output base to x~
K => Execute learn line x times~        | $ => Restart/Reset configuration
M => Set Max digits in a number~        | > => Write configuration file
N => Generate a random Number           | < => Read configuration file
Q => Quit to end the program            | ] => Write command history file
R => Square root of x, Re = Remainder   | [ => Read command history file
S => Square x, matrix multiply          | # => Primorial (like factorial)
@ => Square x, element by element       | ? => Help

~ x is a scalar
```

The infix operators are:

```
A = X + Y  =>  Set A to X plus Y
A = X - Y  =>  Set A to X minus Y
A = X * Y  =>  Set A to X times Y, matrix multiply
A = X @ Y  =>  Set A to X times Y, element by element
A = X / Y  =>  Set A to X divided by Y and set Re to remainder
A = X ^ Y  =>  Set A to X to the power Y
```

```
A = X # Y  =>  Set A to Mag(X, Y) = SqRt(Sq(X) + Sq(Y)) and set Re
A = X % Y  =>  Set A to Mod(X, Y) = X Modulo Y
A = X \ Y  =>  Set A to GCD(X, Y) = Greatest Common Divisor
A = X & Y  =>  Set A to 1 if X and Y are not 0, else set A to 0
A = X | Y  =>  Set A to 1 if X or Y, is not 0, else set A to 0
A = X < Y  =>  Set A to 1 if X < Y, else set A to 0
A = X > Y  =>  Set A to 1 if X > Y, else set A to 0
A = X = Y  =>  Set A to 1 if X = Y, else set A to 0, same as ==
A = X == Y  =>  Set A to 1 if X = Y, else set A to 0
A = X <= Y  =>  Set A to 1 if X <= Y, else set A to 0
A = X != Y  =>  Set A to 1 if X not = Y, else set A to 0
A = X <> Y  =>  Set A to 1 if X not = Y, else set A to 0, same as !=
A = X >= Y  =>  Set A to 1 if X >= Y, else set A to 0
A = X -- Y  =>  Set A to X * Repunit(Y), 3--5 = 33333, five threes
```

The assignment operators are:

```
=  =>  Set Top to Top (no-op)
X =  =>  Bring X to top of list
= Y  =>  Set Top to Y
+= Y  =>  Set Top to Top plus Y
-= Y  =>  Set Top to Top minus Y
*= Y  =>  Set Top to Top times Y, matrix multiply
@= Y  =>  Set Top to Top times Y, element by element
/= Y  =>  Set Top to Top divided by Y and set Re to remainder
%= Y  =>  Set Top to Mod(Top, Y) = Top Modulo Y
X = Y  =>  Set X to Y
X += Y  =>  Set X to X plus Y
X -= Y  =>  Set X to X minus Y
X *= Y  =>  Set X to X times Y, matrix multiply
X @= Y  =>  Set X to X times Y, element by element
X /= Y  =>  Set X to X divided by Y and set Re to remainder
X %= Y  =>  Set X to Mod(X, Y) = X Modulo Y
```

The procedures supported are:

```
     AllD(X) => Compute all divisors of X~
AllowFHT(X) => Set Allow FHT multiple on if X != 0, else off~
AutoDisplay(X) => Set Auto display on if X != 0, else off~
     Base(X) => Set Input and Output base to X, [2, 36]~
    BaseI(X) => Set Input base to X, [2, 36]~
    BaseO(X) => Set Output base to X, [2, 36]~
   D1Line(X) => Set display numbers on a single line on if X != 0
    DBool(X) => Set display Boolean state (False/True) if X != 0
     DDig(X) => Set display number of digits in a numbers on if X != 0
    BernM    => Returns the Max index of saved Bernoulli numbers
    ChDir(F) => Change directory to F = "ccc...c", F optional
 ClearBern    => Clear storage of saved Bernoulli numbers
ClearEuler    => Clear storage of saved Euler numbers
 ClearHist    => Clear history of previous operator entries
  ClearLog    => Clear the log file
     Diag(X) => Set diagnostic mode on or off, (X) is optional~
   EulerM    => Returns the Max index of saved Euler numbers
     Exit    => Totally quit the program with no questions asked
 ForceFHT(X) => Set Force FHT multiple on if X != 0, else off~
  GenBern(X) => Generate and save Bernoulli number upto B(X)~
 GenEuler(X) => Generate and save Euler number upto E(X)~
    HelpH(X) => Set Height of Help form in pixels~
    HelpW(X) => Set Width of Help form in pixels~
    HistH(X) => Set Height of History form in pixels~
    HistW(X) => Set Width of History form in pixels~
```

```
LogScreen(X) => Log screen to log file mode, on or off~
 LX => LT    => Restore LastTop to top of the list
     Next    => Move to next item on the list (no argument)
    Pause    => Pause the calculations to free the processor
      PFA(X) => Run prime factor algorithm A on X~
      PFB(X) => Run prime factor algorithm B on X~
      PFE(X) => Run prime factor algorithm ECM on X (fastest)~
     PTab(X) => Write prime table to XJCalcPTab.txt, X primes~
      Pri(X) => Set the execution priority in the operating system~
    Quiet(X) => Set the quiet mode on or off, (X) is optional~
      Ran    => Randomly start a new random number sequence
    ReadN(F) => Read number from file, F = "ccc...c" optional
  Restore    => Restore Configuration, History, & List
      Run(F) => Run XJCalc code from file F, F is optional
     RunH(X) => Set Height of Run form in pixels~
     RunW(X) => Set Width of Run form in pixels~
     Save    => Save Configuration, History, & List
  SaveTop(X) => Set "save top value in LastTop" on or off~
     SetC(X) => Set max commands in history~
     SetD(X) => Set max decimal digits in display~
     SetM(X) => Set max decimal digits allowed in a number~
   SetMax(X) => Set max decimal digits allowed in a number~
     Time    => Set timing mode on without other diags
    Write(X) => Output X, (X may be "ccc...c", X is optional)
  WriteLn(X) => Write(X) and a line feed
   WriteN(F) => Write X to file F = "ccc...c", F is optional
    XJCIn(F) => Enter file name F = "ccc...c" for J command
   XJLOut(F) => Enter file name F = "ccc...c" for H command
    XJNIn(F) => Enter file name F = "ccc...c" for I command
   XJNOut(F) => Enter file name F = "ccc...c" for W command

Note: Top is the item currently on top of the named number list.
~ x is a scalar
```

The functions supported are:

```
      Abs(X) = AbsoluteValue(X)
    BernD(X) = Denominator of Bernoulli number B(X)
    BernN(X) = Numerator of Bernoulli number B(X)
  Bino(X, Y) = Binomial coefficient (X, Y) by binary splitting
 BinoS(X, Y) = Binomial coefficient (X, Y) by standard method
  Chin(X, Y) = add to Chinese remainder problem z == X~ mod Y~
 Chin1(X, Y) = Initialize Chinese remainder with X1~, Y1~
Concat(X, Y) = Concatenate the columns of matrix X and Y
ConcatR(X, Y) = Concatenate the rows of matrix X and Y
     CuRt(X) = CubeRoot(X), Re = Remainder
   Del(M, C) = Delete column C~ of matrix M
  DelR(M, R) = Delete row R~ of matrix M
      Det(M) = Determinate of matrix M
Diagonal(V) = Diagonal matrix from a column or row vector V
  Dig(X, Y) = Number of base Y digits in X, Y >= 2
    DigD(X) = Number of decimal digits in X
 DivInt(X, Y) = Floor(X/Y), integer divide, e by e
 DivRem(X, Y) = Floor(X/Y) and set Re to remainder, e by e
    Euler(X) = Euler number E(X)
   EulerN(X) = Numerator of E(X) = E(X)
Extract(M, C) = Extract column C~ from matrix M
ExtractR(M, R) = Extract row R~ from matrix M
     Fac(X) = Factorial of X by binary splitting
  Fac2(X, Y) = Fac(X) / Fac(Y) by binary splitting
    FacM(X) = (Factorial of X) Mod FMB by binary splitting
 FacM2(X, Y) = Fac(X) / Fac(Y) Mod FMB by binary splitting
```

```
        FacMS(X) = (Factorial of X) Mod FMB by standard method
         FacS(X) = Factorial of X by standard method
          Fib(X) = FibonacciNumber(X), Fib(0) = 0
          Gam(X) = GammaFunction(X) = (X-1)! by binary splitting
       GCD(X, Y) = Greatest Common Divisor X, Y
      GCDe(X, Y) = Extended GCD(X, Y) = X*X1 + Y*Y1
   Get(M, R, C) = Returns element at row R~, column C~ of matrix M
Insert(M, V, C) = Insert column vector V at column C~ of matrix M
InsertR(M, V, R) = Insert row vector V at row R~ of matrix M
      Inv(X, Y) = Z = Inverse of X Mod Y, X*Z == 1 Mod Y
      IsDiag(M) = Returns 1 if M is a diagonal matrix, else 0
        IsSq(X) = 1 (True) if X is a square, else 0
     Kron(X, Y) = Kronecker-Legendre symbol X over Y
      LCM(X, Y) = Least common multiple X, Y
      Mag(X, Y) = SqRt(Sq(X), Sq(Y)), Re = Remainder
      Mat(R, C) = Return a zero matrix with R~ rows and C~ columns
       MatId(X) = Equals an X~ by X~ identity matrix
      Max(X, Y) = Greater of X and Y
         MEq(X) = Mersenne equation = 2^X - 1
      Min(X, Y) = Lesser of X and Y
      Mod(X, Y) = Remainder of X/Y
     Mord(A, N) = Multiplicative order of base A (mod N) or 0
         MPG(X) = The X'th Mersenne Prime Generator, MPG(1) = 3
         MPP(X) = Mersenne Prime Power, MPP(1) = 2
      MPrime(X) = 1 (True) if 2^X - 1 is a Mersenne Prime else 0
          Mu(X) = Moebius Mu(X) function
           P(X) = The X'th prime
      Pascal(X) = Equals an X by X~ Pascal triangle Pij = Bino(i,j)
         PEq(X) = Perfect equation = (2^X - 1) * 2^(X-1)
         PGT(X) = First prime > X
         Phi(X) = Euler's totient function
     PhiL(X, A) = Legendre's formula
          Pi(X) = Number of primes <= X by sieve or Lehmer
         PiL(X) = number of primes <= X by Lehmer's formula
        PiL1(X) = number of primes <= X by Legendre's formula
         PiM(X) = number of primes <= X by Meissel's formula
         PLT(X) = Largest prime < X
         PNG(X) = The X'th Perfect Number Generator, PNG(1) = 6
      Pow(X, Y) = X to the Y
     PowM(X, Y) = (X to the Y) Mod FMB
       Prime(X) = 1 (True) if X is Prime else 0
       Primo(X) = Primorial, product of all primes <= X
       PrimR(X) = Largest and smallest primitive root of X or 0
      PrimRP(X) = Largest and smallest prime primitive root of X
   PrimRQ(X, Y) = 1 (True) if X is a primitive root of Y, else 0
         Rev(X) = Digit Reversal of X base 10
      Rev(X, Y) = Digit Reversal of X base Y
     RInt(X, Y) = RandomInteger between X and Y inclusive
          RN(X) = RandomNumber(X=Seed or matrix)
          SE(X) = X Squared, element by element
Set(M, R, C, Y) = Return M with element at row R~, col C~ set to Y~
   SetAll(M, Y) = Return matrix M with all elements set to Y~
         Sig(X) = Sum of divisors of X
        Sig0(X) = Sum of divisors of X (-X)
        Sign(X) = 0 if X=0, else = X / |X|
        Size(X) = {r, c} where r = rows in X, c = columns in X
 Solve(X, Y, N) = Solve for z, X * z == Y Mod N
     Sord(A, N) = Multiplicative suborder of base A (mod N) or 0
       SortC(X) = Sort each column of matrix X in numerical order
SortCS(X, Y, Z) = Sort each column of matrix X, row Y thru Z
      SortR(X) = Sort each row of matrix X in numerical order
SortRS(X, Y, Z) = Sort each row of matrix X, column Y thru Z
         Sq(X) = X Squared, matrix multiply
```

```
                  SqFree(X) = 1 (True) if X is a squarefree, else 0
                    SqRt(X) = SquareRoot(X), Re = Remainder
                 SumD(X, Y) = Sum of base Y digits in X, Y >= 2
                   SumDD(X) = Sum of decimal digits in X
                      Tau(X) = Number of divisors of X
                   Trace(M) = Sum of elements on the principal diagonal of M
                    Tran(M) = Transpose of matrix M
   {M11, M12, ...; ..., Mrc} = Enter a matrix with r rows and c columns, Mij~
                         -X = Negative of X, 0 - X
                         +X = Positive of X, 0 + X
                         !X = Not X, 0 -> 1 else 0
```

Note that the ' character can be used to separate the arguments in functions since commas are allowed in numeric input, ", " works also.
x~ x is a scalar

```
            +------------Function Keys on Run form------------+
            |   F1  => Display Help form (?)                  |
            |   F2  => Totally Quit/end the program (Q)       |
            |   F3  => Restore previous input command         |
            |   F4  => Restore previous input and accept      |
            |   F5  => Get Status of calculation              |
            |   F6  => Display Configuration form             |
            |   F7  => Display Restore Input History form     |
            |   F8  => Accept input and Calculate             |
            |   F9  => Toggle Logging to Log file on/off (H)  |
            |   F11 => Clear output text box                  |
            |   F12 => Pause (Pause)                          |
            |   Ctrl+F2  => Restart ($)                       |
            |   Ctrl+F9  => Clear Log File (ClearLog)         |
            |   Ctrl+F11 => Clear input text box              |
            |   Ctrl+S   => Save All (Save)                   |
            |   Ctrl+O   => Restore All (Restore)             |
            |   ESC => Clear Run form Input Text Box          |
            |   ESC => Interrupt/Abort a long calculation     |
            |  PgUp => Display previous newer command         |
            |  PgDn => Display previous older command         |
            +-------------------------------------------------+
```

Commands used in XJCalc code files:

```
        If {expression} Then {statements} Else {statements}
        GoTo {label} (label is a name without a colon)
        GoUpTo {label}
        Labels: A name followed by a colon (:)
        Continuation lines ending with + or -
        Batch Commands (Echo, @Echo, Pause, and Rem)
        Echo On/Off
        @Echo On/Off
        Pause
        Rem ... or //... (for remarks)
```

Primitive op codes -

Primitives that act on a number, act on the currently active item. In the following description of primitives, the currently active item is called x for convenience.

A => Auto Display on/off:

Normally, after each command line is executed, the name and numerical value of the currently active item on the list is displayed. When computing with numbers with many significant digits, the time spent in producing this display can be excessively large. It is desirable then to be able to prevent this automatic display. Each time the A command is given the selection status of this option is reversed (toggled).

A word about the displayed value is in order. As an example, if the first command line you enter after starting the program is 1000!, the response will be:

X = 402,38726,00770,93773,54370,24339,23003,98571,93748,64210,71463,25437,99910,
42993,85123,98629,02059,20442,08486,96940,48...,00000,00000,00000,00000,00000,00
000,00000,00000,00000,00000,00000,00000,00000,00000,00000,00000,00000,0000
0,00000 (2568 Digits)

The (2568 Digits) means that integer has 2568 significant decimal digits. The ... means that not all of the middle significant digits are displayed. Use the SetD(X) procedure to control the max digits displayed.

B => Display learn line:

The calculator can contains a learned line, see the X primitive to enter and execute the learned line. The B command displays the current contents of the learned line. After the B command is executed, the F3 and F4 keys will restore the input line to the contents of the learned line instead of the previously typed command line. The B command enters the learned line command into the top of the history list of commands.

C => Change sign of x:

This is the same as multiplying x by minus one. Negative numbers can be entered by preceding them with a minus sign. The x referenced here is the current active item on the list of variables.

F or ! => Factorial by binary splitting:

Replaces x with the factorial of x = 1 * 2 * 3 * ... * x. The standard method computes Fac(n) by

```
x1 = 1, x2 = 2*x1, x3 = 3*x2, ... x(n) = n*x(n-1), Fac(n) = x(n).
```

For the binary splitting method, define the product

```
Pr(a, b) = (b+1)*(b+2)* ... * (a-1)*a = Fac(a) / Fac(b).
```

This is computed by

```
Pr(a, b) = Pr(a, (a+b)/2) * Pr((a+b)/2, b),
```

where the two terms in the product are computed recursively in the same way until a-b is small:

```
d = a-b; m = (a+b)/2 // integer divide
If (d) > 3, Pr(a, b) = Pr(a, m) * Pr(m, b);
Else if (d) == 0, Pr(a, b) = 1;
Else if (d) == 1, Pr(a, b) = a;
Else if (d) == 2, Pr(a, b) = a*(a-1);
Else if (d) == 3, Pr(a, b) = a*(a-1)*(a-2);
```

Else Pr(a, b) = 0.

For the factorial

Fac(n) = Pr(n, 0).

See: http://numbers.computation.free.fr/Constants/Algorithms/splitting.html

G => Set Digits/Group~:

The G command will set the number of digits per group to the current value of x. If this is set to 3, numbers will be displayed with a comma after every 3rd digit like 1,234,567,890. If this is set to zero or a negative number, no commas will be displayed. The ~ indicates that x must be a scalar.

H => Log screen to log file toggle:

The H command causes all output to the screen to be logged to a disk file. See the XJLOut(F) procedure for specifying a file name for this purpose. Each time the H command is given the selection status of this option is reversed/toggled.

The output text box on the Run form is referred to as the output screen or merely as the screen when the context is clear.

I => Input number from file:

The I command will use the last entered comment as a file name and read this file as a XJCalc formatted number and assign it to the current active item. It is assumed that the file was created by the W command or the WriteN procedure.

See the W command for the format of file names. If a comment has not been entered, the file name NoName.XJN is used. The XJNIn procedure can be used to give an override file name. File names can also be assigned using the Configuration form.

The files input by the I command are assumed to have all ASCII text characters with a numerical value less than 128. See the ReadN procedure. The DOS VPCalc program deleted the upper bit of characters greater than 127.

J => Run XJCalc code from file:

The J command will use the last entered comment as a file name and read this file as a text file. Each line of the file will be interpreted as a XJCalc command line and executed. If comment commands and J commands exist in the text file, these other referenced files will be opened and processed. The only limitation to this nesting of code files is the availability of memory and buffers. If a comment has not been entered, the file name NoName.XJC is used. The XJCIn procedure can be used to give an override file name for the J command. The files input by the J command are assumed to have all ASCII text characters with a numerical value less than 128.

K => Execute learn line x times~:

This will cause the learned line to be executed x time. x must be in the range 0 <= x <= 2,147,483,647 (2^31 - 1). The x referenced here is the current active item on the list of variables. If x is larger than this max, then the max will be used. A long repetition of a learned line can always be interrupted by using the ESC key or selecting the Abort Calc. button on the Run form.

M => Set Max digits in a number~:

The M command will set the current value of the maximum number of decimal digits allowed in an integer to the current value of x. If there are items on the list containing more than this number of digits, they will be reduced to contain at most this number of digits and the most significant digits will be lost. If x is not a multiple of 8, when the M command is given, then the next higher multiple of 8 is used. If x is less than 40, it is set to 40.

N => Generate a random number:

The N command generates a random number between zero and 10^35 and assigns it to the current active item on the list. This number will never have more than 35 significant decimal digits. Theoretically the random number generator will cycle after 10^35 numbers, but the earth will not last that long. The items RN, RNA, and RNC are put on the list by the random number command. The equation used is: x = RN = (RNA * RN + RNC) mod (10^35), where RNA and RNC are 35 digit integers. RNA and RNC are added to the list so they can be examined.

Q => To totally Quit/end the program:

The program exits after displaying a message block to allow the operator to OK or Cancel the request.

R => Square root of x, Re = Remainder:

x is replaced with the positive square root of x, error if x < 0. The remainder, Re = x - Sq(SqRt(x)), is also computed and added to the list.

S => Square x, matrix multiply:

x is replaced with the square of x, matrix multiply.

@ => Square x, element by element:

x is replaced with the square of x, element by element.

W => Write x (Top number) to file:

The W command will use the last entered comment as a file name and write register x into this file as a XJCalc formatted number. This number can be reread into x by the I command or ReadN procedure. If the file already exists, it will be erased and recreated. For example, the following are valid file names:

"File.Ext" File.Ext is on the default directory, "C:\Direct\File.Ext" File.Ext is on C: drive, Direct directory.

If the file name does not have a period, the extension .XJN is added. If a comment has not been entered, the file name NoName.XJN is used. The XJNOut procedure can be used to give an override file name for the W command. The file written is a text file and can easily be browsed and read by other programs. The content of the file for 100! is:

{1 blank line}

```
X =

933,26215,44394,41526,81699,23885,62667,00490,71596,82643,81
621,46859,29638,95217,59999,32299,15608,94146,39761,56518,28
625,36979,20827,22375,82511,85210,91686,40000,00000,00000,00
000,00000
(158 Digits)
```

The content of the file for a base 36 random number is:

```
{1 blank line}
Base(36)

RN =

0GP,69HZZ,40JSW,YWIHX,KAAND
(22 Digits Base 36)
```

X => Learn, Execute (X learn, X):

If this is the last command on a command line, then it caused the learned line to be executed once. If not the last command on the line, this command stores all the commands following on the same line or text box as this one into the learned line. Execution of the current line is stopped. Type the learned line:

```
N=0 Fact=1 X N=N+1 Fact=Fact*N Z X
```

and then do an X commands. You might want to key in an H command before the X command to turn on logging. This will print a table of factorials from 2! to (2147483647)! or so, if you wait long enough. Hit the ESC key to interrupt and abort the operation if you get tired of waiting. After the X command is executed, the F3 and F4 keys will restore the input line to the contents of the learned line instead of the previously typed command line.

Y => Delete (Yank) number from list:

The Y command removes the currently active item from the list and makes the next older item the active item. The age of an item is judged by when it was created. X is always the oldest item and is never removed from the list. If the Y command is executed, when X is the active item, X is not removed, but the youngest item becomes the active item. Thus, a long string of Y commands will always remove all items from the list except X.

Z => Output the list of variables:

The Z command will display the name and value of all items on the list. Some items may be found on the list that were not explicitly put there. The items RN, RNA, and RNC are put on the list by the random number command N and the random number function RN(X). The item File: "comment" is put on the list by the "comment" command. The item Lrn: {learned line} is put on the list by the X command. The items File: "comment" and the item Lrn: {learned line} also have a value associated with them, normally = 0. This value has no meaning and is not used. The item Re is put on the list by the / and # infix operator, and by the CuRt, DivRem, Mag, and SqRt functions.

If diags are turned on by Diag(1) when the Z command is executed, extra items are displayed. For example the output:

```
1: X = 0 N=1 M=1 U=1
```

says that item 1: on the list is X = 0, it has N=1 super-digits, memory for M=1 super-digits, and the upper bound of the array for its super-digits U=1.

" => Start/Stop file name or comment:

Comments can be entered anywhere on the command line. The comment is started with a " mark. The comment is ended with a " mark or the end of the line. All blank spaces between the " marks become part of the comment. Comments are also used as file names; see the XJCIn, XJNIn, XJNOut, and XJLOut procedures. The item File: {comment} is put on the list by this "{comment}" command.

% => Set FMB = x, FMB on list:

The % primitive command is equivalent to FMB = x, where x is the currently active item. FMB stands for Floating Modulo Base. The / primitive command and the PowM(X, Y) function use FMB from the list. Normally FMB > 0, but it can be < 0 to give negative residuals.

All modulo and division operations are performed to make the remainder of the division to have the same sign as the divisor or be zero. For q = x/y with remainder r (y != 0), 0 <= |r| < |y| and Sign(r) = Sign(y) or r = 0, and x = q*y + r. For example:

```
14/5 = 2, r = 4,
14/(-5) = -3, r = -1,
(-14)/5 = -3, r = 1,
(-14)/(-5) = 2, r = -4.
```

/ => x = x Mod FMB, FMB on list:

The / primitive command replaces x with x modulo FMB, where x is the currently active item and FMB is an item on the list. If FMB is not on the list, it is added to the list with a value of zero. If FMB is zero, the value of x is not changed, else 0 <= |x| < |FMB| and Sign(x) = Sign(FMB) or x = 0.

\ => Set Input and Output base to x~, [2, 36]:

The primitive command \ sets both the input and output base to x. This is the same as the Base(X) procedure described more fully below.

$ => Restart/Reset configuration:

This reinitializes the program, the same as reloading except total running time is not reset, the history of previous operator entries is not cleared, and the log screen to log file state is not changed. The parameter set by the M command is reset to its max value, and all items on the list are deleted except X and it is cleared. This also reset the random number generator and FMB is effectively zero.

> => Write configuration to file Config.XJC:

The file Config.XJC is written to disk. It contains the XJCalc commands that will restore the configuration of XJCalc to its current state.

An example of the contents of a Config.XJC is:

```
@Echo Off
```

```
Rem  Start of file Config.XJC
BaseI(0)
SetMax(134218400)
LastTop=;  134218400 M;  5 G;
SetC(1000)
SetD(200)
SaveTop(1)
AllowFHT(1)
ForceFHT(0)
Diag(0)
Time
AutoDisplay(1)
ChDir("C:\ProgramD\VC#\Harry\XJCalc\bin\Debug")
ReadN("FMB.XJN")
XJCIn("NONAME.XJC")
XJLOut("NoName.XJL")
XJNIn("10000!.XJN")
XJNOut("NoName.XJN")
LogScreen(0)
BaseO(8)
BaseI(16)
X=
Quiet(1)
Rem  End of file Config.XJC
Echo On
```

< => Read configuration from file Config.XJC:

The file Config.XJC is read and run as a XJCalc code file. This will restore the configuration of XJCalc to its configuration when the file was written by the > command.

] => Write entry command history to file XJCalcHist.txt:

The file XJCalcHist.txt is written to disk. This is a text file and contains a copy of the current history of operator entries.

[=> Read entry command history from file XJCalcHist.txt:

The file XJCalcHist.txt is read and used to restore the history of operator entries with the history when the file was written by the] command. The current history is not cleared, but some or all of it may be lost since only "Max Commands in History" entries are saved. Duplicate commands are deleted as the new copy is entered.

=> Primorial (like factorial):

Replaces x with the primorial of x. See the Primo(X) function.

? => Display Help form as presented above.

Infix operators -

Infix operators +, -, *, /, ^, %, \, &, |, <, =, >, <=, !=, <>, >=, and -- are the operators that appear between operands in an expression. Infix operators do not change the value of their operands, but produce a single result that can be used to further complete the evaluation of the expression that contains the

infix operator. The infix operator precedence classes, from highest to lowest, are:

```
1) ^
2) *, /, #, %, \, &
3) +, -, |, --
4) <, >, =, ==, <=, !=, <>, >=
```

Operators of the same class are evaluated from left to right. Thus (2 * 10)^2 = 20^2, but 2 * 10^2 = 2 * 100. Also, A + B * C = A + (B * C).

A = X + Y => Set A to X plus Y:

Addition operator. Most calculations with two or more parameters allow the parameters to be mixed, scalars and/or matrices.

A = X - Y => Set A to X minus Y:

Subtraction operator.

A = X * Y => Set A to X times Y, matrix multiply:

Matrix multiplication operator.

A = X @ Y => Set A to X times Y, element by element:

Element by element multiplication operator.

A = X / Y => Set A to X divided by Y and set Re to remainder:

Division operator. The remainder is also computed and added to the list as an item names Re. An error message is given if y = 0. x = a * y + Re. The remainder always has the same sign as y or equal to 0, 0 <= |Re| < |y|. Be careful, x=-13/5 is not (-13)/5 but -(13/5) = -2 with a remainder is +3. (-13)/5 = -3 with a remainder of 2.

A = X ^ Y => Set A to X to the power Y:

Exponential operator. The peasants' method is used in which up to 2 * Log base 2 of y multiplies of powers of x are done to compute the result. An error message is generated if x = 0 and y is < 0. If x = 0 and y = 0, an answer of 1 will be given. The ^ operator is evaluated from right to left: 3^3^3 = 3^27 = 762,55974,84987. My MS Dos program VPCalc evaluated it as (3^3)^3 = 27^3 = 19683, which is not the normal convention.

A = X # Y => Set A to Mag(X, Y) = SqRt(Sq(X) + Sq(Y)) and set Re:

Sets A to the integer squareroot of X*X + Y*Y and set the item Re equal to the remainder of the squareroot.

A = X % Y => Set A to Mod(X, Y) = X Modulo Y:

Modulo operator. a = x % y = x Mod y, 0 <= |a| < |y| and Sign(a) = Sign(y) or a = 0. An error message is generated if y = 0.

A = X \ Y => Set A to GCD(X, Y) = Greatest Common Divisor:

Greatest common divisor operator. Uses the oldest algorithm in the book, Euclid's algorithm (see Euclid's Elements, Book 7, Propositions 1 and 2). Only the absolute values of x and y are used in the computation. For example, the GCD of 12 and -18 is 6.

A = X & Y => Set A to 1 if X and Y are both not 0, else set A to 0:

Logical And operator. For all logical operations, 0 is considered False and all other values are considered True. When the result of a logical operation is True, the value 1 will be produced. When the result of a logical operation is False, the value 0 will be produced.

A = X | Y => Set A to 1 if X or Y, is not 0, else set A to 0:

Logical Or operator.

A = X < Y => Set A to 1 if X < Y, else set A to 0:

Numerical Less-than operator. For all numerical equivalence operators, the operands are considered as signed integers and the result is either 1 (True) or 0 (False).

A = X > Y => Set A to 1 if X > Y, else set A to 0:

Numerical Greater-than operator.

A = X = Y => Set A to 1 if X = Y, else set A to 0, same as ==:

Numerical Equal-to operator.

A = X == Y => Set A to 1 if X = Y, else set A to 0:

Numerical Equal-to operator.

A = X <= Y => Set A to 1 if X <= Y, else set A to 0:

Numerical Less-than-or-equal-to operator.

A = X != Y => Set A to 1 if X not = Y, else set A to 0:

Numerical Not-equal-to operator.

A = X <> Y => Set A to 1 if X not = Y, else set A to 0, same as !=:

Numerical Not-equal-to operator.

A = X >= Y => Set A to 1 if X >= Y, else set A to 0:

Numerical Greater-than-or-equal-to operator.

A = X -- Y => Set A to X * Repunit(Y), 3--5 = 33333, five threes:

A repunit is a number consisting of copies of the single digit 1. Repunit(n) base B is (B^n -1)/(B - 1) for base 10 this is (10^n - 1)/9. The output base (BaseO) is used for this calculation. This allows you to have one value for the base of the input numbers and another value for the repunit base.

Assignment operators -

= => Set Top to Top (no-op):

Same as Top = Top, a no-op.

X = => Bring X to top of list:

The item X is made the current active item.

= Y => Set Top to Y:

Same as Top = Y. Evaluate Y and store it on the list as Top, where Top is the current active item on top of the list.

+= Y => Set Top to Top plus Y:

Same as Top = Top + (Y). Evaluate Y and add it to Top, where Top is the current active item on top of the list.

-= Y => Set Top to Top minus Y:

Same as Top = Top - (Y).

*= Y => Set Top to Top times Y, matrix multiply:

Same as Top = Top * (Y).

@= Y => Set Top to Top times Y, element by element:

Same as Top = Top @ (Y).

/= Y => Set Top to Top divided by Y and set Re to remainder:

Same as Top = Top / (Y).

%= Y => Set Top to Mod(Top, Y) = Top Modulo Y:

Same as Top = Top % (Y).

X = Y => Set X to Y:

Basic assignment operator, evaluate Y and store it on the list as X.

X += Y => Set X to X plus Y:

Same as X = X + (Y). Evaluate Y and add it to X.

X -= Y => Set X to X minus Y:

Same as X = X - (Y).

X *= Y => Set X to Mod(X, Y) = X Modulo Y , matrix multiply:

Same as X = X * (Y).

X @= Y => Set X to X times Y, element by element:

Same as X = X @ (Y).

X /= Y => Set X to X divided by Y and set Re to remainder:

Same as X = X / (Y).

X %= Y => Set X to Mod(X, Y) = X Modulo Y:

Same as X = X % (Y).

Procedures -

Procedures are invoked by a statement starting with a procedure name followed by its argument. Arguments are numerical expressions that are evaluated before the procedure is performed. Procedures do not change the value of their arguments. For the procedures Write and WriteLn, arguments are optional and may be literal like: WriteLn("Now is the time"). For some procedures like Next, arguments are not allowed.

AllD(X) => Compute all divisors of X~~:

Uses the prime factor algorithm A to factor x and then compute all of the positive integral divisors of x. For example, if x = 12 the output is:

      All 6 Divisors: 1; 2; 4; 3; 6; 12.

The first divisor is always 1 and the last divisor is always x. The number of divisors is displayed in decimal but the divisors are displayed in the current output base.

AllowFHT(X) => Set Allow FHT multiple on if X != 0, else off~:

The allow FHT mode is turned on if x != 0 and is turned off if x = 0. When this mode is on, long multiplications are speeded up by using the fast Hartley transform method to do the convolution. FHT is used if the numbers are greater than about 236 decimal digits. For about 100,000 digits numbers the FHT multiply runs in 0.1% of the time of a normal multiply (a factor of 1000). This mode is initially on.

      For fast Hartley transform multiply

fxt subroutines converted from C++ to C#
including FHT Convolution with zero padded date
by Harry J. Smith.

C++ author = Joerg Arndt email: arndt@jjj.de
the C++ software is online at http://www.jjj.de/

---------- *** LEGAL NOTICE: *** ----------

---------- *** end of legal notice *** ----------

Also, when this mode is on, divides and square roots are speeded up by using Newton-Raphson iterations. For u/d, 1/d is computed first. No divides are performed except one at low precision to get the first guess. This is referred to as "divisionless divide".

x = 1/d: x = (x + x) - (x * x) * d

For SqRt(s), there is a long divide each iteration, but this is performed by the divisionless divide. The divide by 2 is a fast short division.

x = SqRt(s): x = (x + s/x)/2

In both cases, the precision used to calculate x is doubled with each iteration.

AutoDisplay(X) => Set Auto display on if X != 0, else off~:

Same as the A primitive op code, but instead of being a toggle, sets Auto display on if x != 0, and sets it off if x = 0.

Base(X) => Set Input and Output base to X, [2, 36]~:

There are two configuration items, BaseI and BaseO, that determine the numerical base of input numbers and output numbers. This command sets both of them to x. If x is less than 2, they are set to 10. If larger than 36, they are set to 36. The letter A through Z are used to represent digits larger than 9. A represents 10 decimal and Z represents 35 decimal. This gives hexadecimal numbers their normal representation.

All the numbers in input commands are affected by the input base, so the command Base(10) is always a no-op. To change the base to 10 decimal use the command Base(0).

Not all output digits are displayed in BaseO, only the numerical value of numbers on the list. When BaseI is not 10 decimal, the command prompt will contain the input base expressed in decimal, and when BaseO is not 10 decimal, the displayed numbers will have the output base displayed in decimal. For example, if both BaseI and BaseO = 16 decimal, the input of X=0FF would get the response:

Command (Base 16): X=0FF

X = 0FF (Base 16)

The leading zero in the input 0FF was needed to distinguish it from the primitive F command. In general, if a number starts with a digit larger than 9, precede it with a zero (0).

BaseI(X) => Set Input base to X, [2, 36]~:

This is like the Base(X) command, but only the Input base is affected.

BaseO(X) => Set Output base to X, [2, 36]~:

This is like the Base(X) command, but only the Output base is affected.

D1Line(X) => Set display numbers on a single line on if X != 0, else off.

DBool(X) => Set display Boolean state (False/True) if X != 0, else off.

DDig(X) => Set display number of digits in a numbers on if X != 0, else off.

BernM => Returns the Max index of saved Bernoulli numbers:

When Bernoulli numbers are generated, the exact numerator and denominator of all even indexed Bernoulli numbers up to the maximum index generated are saved. This command returns this maximum index currently in storage.

ChDir(F) => Change directory to F = "ccc...c", F optional:

Changes the directory used for commands H, I, J, W, ReadN(F), and Run(F). The original directory when the program starts is called the Home directory and is always used for the commands >, <,], [, ?, ClearHist, ClearLog, LogScreen, Restore, and Save, commands. A ChDir without F will not change the directory, but will tell you how it is currently set.

It is probably easier to use the Configuration form “Change File Path” command button to change the disk directory. This procedure was included so it could be used by the Config.XJC code file for the >, <, Save, and Restore commands.

ClearBern => Clear storage of saved Bernoulli numbers.

ClearEuler => Clear storage of saved Euler numbers.

ClearHist => Clear history of previous operator entries:

The history of up to “Max Commands in History” previous operator entries are saved and can be retrieved by selecting the “Restore Input” button on the Run form. The ClearHist procedure removes all operator entries currently saved and makes this memory available to the calculator. Even though no argument is needed for this and some other procedures, it is usually better to use the parentheses, e.g., ClearHist() or ClearHist(to prevent unexpected results if the procedure name is misspelled.

ClearLog => Clear the log file:

If the log file is open, the file is closed and reopened. If it is currently closed, it opened and then closed. In either case it is cleared. Initially the log file name is NoName.XJL.

The cleared log file will have up to three lines of data like:

```
Log file NoName.XJL Cleared 10/6/2007 1:08:10 PM
XJCalc - Extra Precision Integer Matrix Calculator, C# Version 3.2.1.26037
Run on: Harry's Intel 3 GHz Pentium 4 - Dell DGV4T641 - Windows XP Pro SP2
```

The third line is generated by having something like:

```
SET SYSTEM=Harry's Intel 3 GHz Pentium 4 - Dell DGV4T641 - Windows XP Pro SP2
```

in your AutoExec.Bat file.

Diag(X) => Set diagnostic mode on or off~:

The diagnostic mode is turned on if x != 0 and is turned off if x = 0. When the diagnostic mode is on, all command line executions will be timed by the computer clock and the time spent executing the command will be displayed. The timing data is displayed as:

```
T = xxx.xx  DT = xx.xx sec.  Start execution
{command output, if any}
T = xxx.xx  DT = xx.xx sec.  End of execution
```

The DT value on the End of execution line is the time spent executing the command. The DT on the Start execution line is the time spent waiting for the operator to compose the command. The T values are the total running time since the program was started and can only be reset by terminating and reentering the program. The Quit button followed by the Run button will do it.

To turn diags on Diag(1) can be shortened to Diag. If the argument is < 0 like diag(-1) the debug mode is also turn on and even more diagnostic (debug) messages will be displayed.

EulerM => Returns the Max index of saved Euler numbers:

When Euler numbers are generated, the exact integer value of all even indexed Euler numbers up to the maximum index generated are saved. This command returns this maximum index currently in storage.

Exit => Totally quit the program with no questions asked.

ForceFHT(X) => Set Force FHT multiple on if X != 0, else off~:

The force FHT mode is turned on if x != 0 and is turned off if x = 0. When this mode is on, all multiplications are done using the fast Hartley transform method to do the convolution. This mode is initially off.

If ForceFHT is turned on when AllowFHT is off, AllowFHT is turned on also. If AllowFHT is turned off when ForceFHT is turned on, ForceFHT is turned off also.

GenBern(X) => Generate and save Bernoulli number upto B(X)~:

The exact numerator and denominator of all even indexed Bernoulli numbers up to B(x) are generated and saved in computer storage. If x is less than 4, it is taken to be 4. If x is an odd positive integer, the next even integer is used. If x is very large, an error message is generated: “GenBern: n > 10000000, too

large for Bernoulli number". But it would take "forever" for an x = 10,000,000. X must be a scalar.

GenEuler(X) => Generate and save Euler number upto E(X)~:

The exact integer value of all even indexed Euler numbers up to E(x) are generated and saved in computer storage. If x is less than 4, it is taken to be 4. If x is an odd positive integer, the next even integer is used. If x is very large, an error message is generated: "GenEuler: n > 10000000, too large for Euler number". But it would take "forever" for an x = 10,000,000. X must be a scalar.

HelpH(X) => Set Height of Help form in pixels~:

It is handy to put these commands in the AutoExec.XJC file. For example, the lines:

```
HelpW(674) HelpH(900)
RunW(674) RunH(950)
```

Will set the forms to larger than the default size for a 1280 by 1024 screen resolution.

HelpW(X) => Set Width of Help form in pixels~.

HistH(X) => Set Height of History form in pixels~.

HistW(X) => Set Width of History form in pixels~.

LogScreen(X) => Log screen to Log file, on or off~:

Same as the H primitive op code, but instead of being a toggle, sets Log screen to Log file on if x != 0, and sets it off if x = 0. This procedure was called EchoScreen(X) in my VPCalc DOS program, and that command still works.

LX => LT => Restore LastTop to top of the list:

This sets the current active item equal to the value of the item named LastTop. If LastTop does not exist, it is created with a value of zero. Normally, before each command line is executed the value of the current active item is saved on the list in an item named LastTop. If a command line is entered that changes the value of the current active item, it can be restored to its previous value if the LX or LT procedure is performed immediately. This procedure should be entered as a command by itself to prevent LastTop from being changed before it is retrieved.

The value of the current active item, Top, is not saved in LastTop if: 1) The command line is: LX 2) The command line is: LT 3) The command line is: LastTop= 4) The SaveTop option is turned off by SaveTop(0).

Next => Move to next item on the list (no argument):

This changes which item on the list is the active item from the current active item to the next item from top to bottom. If X is the active item, which is always at the bottom of the list, the top item will become the active item. A

command line with the single command Next followed by several F4 function keys will move through the whole list one item at a time. Note, this procedure does not take an argument.

Pause => Pause the calculations to free the processor. See function key F12:

Pause is actually a batch command. All commands on the line after Pause are ignored.

PFA(X) => Run prime factor algorithm A on X~:

Computes the prime factorization of x using algorithm A. For a command of PFA(60) the output is:

```
60 =
2^2 * 3^1 * 5^1.
```

If the input base is 10 and the output base is 16, for a command of PFA(2^15*13^1) the output is:

```
68000 (Base 16) =
2^15 * 0D^1.
```

The number and the prime divisors are displayed in the current output base. The exponents are always displayed in decimal

Algorithm A uses an implementation of a method described in Knuth, Vol. 2, Seminumerical Algorithms, Page 348 for the factorization.

PFB(X) => Run prime factor algorithm B on X~:

Computes the prime factorization of x using algorithm B. The output is the same as PFA(X). Algorithm B uses Algorithm 357 of the Collected Algorithms from ACM, An Efficient Prime Number Generator using the sieve of Eratosthenes, to generate consecutive primes and find the prime divisors of x.

PFE(X) => Run prime factor algorithm ECM on X (fastest)~:

Computes the prime factorization of x using the Elliptic Curve Method (ECM) algorithm. The output is the same as PFA(X).

PTab(X) => Write prime table to XJCalcPTab.txt, X primes~:

Writes a table of x primes to file XJCalcPTab.txt. If XJCalcPTab.txt already exists it is renamed XJCalcPTab.Bak before the new file is written. For example, a command of PTab(100) generates a file containing:

```
XJCalc - Prime Number Table - 10/09/2004 11:45:20 AM
First column is prime index the rest are consecutive primes

1 2 3 5 7 11 13 17 19 23
10 29 31 37 41 43 47 53 59 61 67
20 71 73 79 83 89 97 101 103 107 109
30 113 127 131 137 139 149 151 157 163 167
40 173 179 181 191 193 197 199 211 223 227
50 229 233 239 241 251 257 263 269 271 277
60 281 283 293 307 311 313 317 331 337 347
70 349 353 359 367 373 379 383 389 397 401
```

```
      80 409 419 421 431 433 439 443 449 457 461
      90 463 467 479 487 491 499 503 509 521 523
      100 541
      End of file XJCalcPTab.txt
```

All of the numbers in the table are always in decimal.

Pri(X) => Set the execution priority in the operating system~:

The execution priority is set according to x:

```
      x = 2 => Highest
      x = 1 => AboveNormal
      x = 0 => Normal
      x = -1 => BelowNormal
      x = -2 => Lowest
      Else => Normal;
```

The Pri command without the (X) will display the current priority.

Quiet(X) => Set the quiet mode on or off~:

The quiet mode is turned on if x != 0 and is turned off if x = 0. When the quiet mode is on, some of the status messages all not displayed. The (X) is optional, Quiet, Quiet(1, and Quiet(1) are interpreted as an on command.

Ran => Randomly start a new random number sequence:

This is like then Rn(X) function, but instead of the user supplying a seed; a seed is automatically generated from the current date and time.

ReadN(F) => Read file F = "ccc...c", F is optional:

This will use the argument F = "ccc...c" as a file name and read this file as a XJCalc formatted number and assign it to the item with the name stored in the file. This is the name it had when it was written. It is assumed that the file was created by the W command or the WriteN(F) procedure. See the W command for the format of file names. If no argument is given, and a comment has not been entered, the file name NoName.XJN is used. The files input by the ReadN command are assumed to have all ASCII text characters with a numerical value less than 128. The ReadN proc is similar to the I command. The ReadN command will not change the current active item unless an = sign is not found in the file or the name found is the same as the current active item.

Restore/Save => Restore or Save Configuration, History, & List:

Save will write the entry history file XJCalcHist.txt like the] command, write the configuration file Config.XJC like the > command, write each items on the list to a separate file (Save0000.XJN, Save0001.XJN, ...), and write a XJCalc code file Restore.XJC that can be run by XJCalc to restore all of the saved items.

Restore will read the entry history file XJCalcHist.txt like the [command, run the configuration file Config.XJC like the < command, and run the Restore.XJC restore file to read in each items that was on the list at save time. Restore does not clear the command history or the list before it executes, so they may grow larger than they were at save time.

Run(F) => Run XJCalc code from file F, F is optional:

This will use the argument F = "ccc...c" as a file name and read and run this file as a XJCalc code file. If no argument is given, defaults are like ReadN(F). To see examples of how XJCalc primitives, procedures, and functions are used, inspect the delivered *.XJC files. It will be noted that they are in plain text.

RunH(X) => Set Height of Run form in pixels~.

RunW(X) => Set Width of Run form in pixels~.

Save => Save Configuration, History, & List:

Same as Ctrl+S, Save All.

SaveTop(X) => Set "save top value in LastTop" on or off~:

This sets the "save top value in LastTop" option on if x != 0, and sets it off if x = 0.

SetC(X) => Set max commands in history [1, 100000]~:

The SetC(X) procedure sets the maximum number of commands that will be saved in the command history list. If X evaluates to a number x Greater than 100000, the max commands is changed to 100000. If x is less than 1, the max commands is not changed and the status message “Max commands in history not changed from {max}” is displayed. This message is also displayed if x is the same as the value already being used.

If the value is actually changed, the message “Max commands in history changed from {old max} to {new max}” is displayed. If there are more than x commands already in history, all but the latest x commands are removed.

SetD(X) => Set max decimal digits in display~:

The SetD(X) procedure sets the maximum number of digits to display to the evaluated value of X. If x is less than 14, the value 14 is used. The values set by the M command is always carried as a multiple of eight (8), but the value set by the SetD(X) procedure can be any integer >= 14.

SetM(X) => Set max decimal digits allowed in a number~:

The SetM(X) procedure sets the max decimal digits allowed in any value to the evaluated value of X. Same as the SetMax(X) command or the M primitive op codes.

SetMax(X) => Set max decimal digits allowed in an integer~:

The SetMax(X) procedure sets the max decimal digits allowed in any value to the evaluated value of X. Same as the SetM(X) command or the M primitive op codes.

Time => Set timing node on without other diags:

This selects the timing information that you get when diags are turned on, but without the other diagnostic messages. Turning diags off will turn timing off also.

Write(X) => Output X, (X may be "ccc...c", X is optional):

The Write(X) procedure outputs the evaluated value of X to the console. The H command and the LogScreen(X) procedure can be used to log this output to the Log file. This procedure is mainly useful in XJCalc code files.

WriteLn(X) => Write(X) and a line feed:

The WriteLn(X) procedure is the same as the Write(X) procedure except that the output generated is followed by an end-of-line indicator.

WriteN(F) => Write X to file F = "ccc...c", F is optional):

This will use the argument F = "ccc...c" as a file name and write the current active item as a XJCalc formatted number exactly like the W command. See the W command for the format of file names. If no argument is given, and a comment has not been entered, the file name NoName.XJN is used.

XJCIn(F) => Enter file name F = "ccc...c" for J command:

This establishes the file name of the XJCalc code file that will be read by the next J command. If the ("{filename}") is missing, the file name input with the last "comment" command will be used. If no comment entered, the name NoName.XJC will be used.

XJLOut(F) => Enter file name F = "ccc...c" for H command:

This establishes the file name of the XJCalc log file that will be opened by the next H command that opens a file. If the ("{filename}") is missing, the file name input with the last "comment" command will be used. If no comment entered, the name NoName.XJL will be used.

If the log file name is changed while it is open, the new name will not be used until logging is turned off and then turned on again. Whenever a log file is opened it is opened for append.

XJNIn(F) => Enter file name F = "ccc...c" for I command:

This establishes the file name of the XJCalc number file that will be read by the next I command. If the ("{filename}") is missing, the file name input with the last "comment" command will be used. If no comment entered, the name NoName.XJN will be used.

XJNOut(F) => Enter file name F = "ccc...c" for W command:

This establishes the file name of the XJCalc number file that will be written by the next W command. If the ("{filename}") is missing, the file name input with the last "comment" command will be used. If no comment entered, the name NoName.XJN will be used.

Functions -

Functions are used on the right hand side of an equation or assignment statement. Functions do not change the value of their arguments, but produce a single result that can be used to further complete the evaluation of the expression that contains the function reference.

If a statement starts with a function reference like a procedure, then the function is evaluated and this value is assigned to the current active item. It is best not to have one of the calculator created items like FMB, LastTop, Re, RN, RNA, or RNC as the current active item when using a function as if it were a procedure.

Abs(X) = AbsoluteValue(X):

Absolute value function = |X|.

BernD(X) = Denominator of B(X):

This function returns the exact denominator of the Bernoulli number B(x), a rational number reduced to its lowest terms. If x < 0, 1 is returned. If x is too large, 1 is returned and an error message is generated.

BernN(X) = Numerator of B(X):

This function returns the exact numerator of the Bernoulli number B(x), a rational number reduced to its lowest terms. If x < 0, 0 is returned. If x is too large, 0 is returned and an error message is generated.

Bino(X, Y) = Binomial coefficient (X, Y) by binary splitting:

This returns the binomial coefficient of x things taken y at a time. If y < 0, Bino(x, y) = 0. If x < 0, Bino(x, y) = (-1)^y * Bino(y-x-1, y). If y > x, Bino(x, y) = 0.

For Bino(a, b) 0 <= b <= a, if b < a/2, set b = a - b. now

Bino(a, b) = Pr(a, b) / Fac(a-b).

BinoS(X, Y) = Binomial coefficient (X, Y) by standard method:

Same as Bino(X) except does not use binary splitting.

Chin(X, Y) = add to Chinese remainder problem z == X~ mod Y~:

This uses the Chinese remainder theorem (CRT) to solve

z == x1 mod y1
z == x2 nod y2

for z, where x1 is the value of X1 and y1 is the value of Y1 on the named number list, x2 is the x and y2 is the y value being input. X1 on the list is set to the solution z and Y1 is set to the new modulus = y1*y2/g. A solution exists if g = GCD of y1 and y1 is equal to one or (y1 - y2) mod g = 0. If no solution exists, z, X1, and Y1 are set to zero. The signs of y1 and y2 are not used.

Note, z == x mod y is to be read z is congruent to x modulo y. This means that z and x have the same remainder when divided by y. This is the same as (z - x) is

divisible by y, or (z - x) mod y = 0. Since y > 0, the mod operator here is the remainder function. x mod y = r such that 0 <= r < y, and x = q * y + r, and q the integer quotient of x / y.

Chin1(X, Y) = Initialize Chinese remainder with X1~, Y1~:

Y1 on the named number list is set to |y| and X1 to x mod Y1. The value X1 is returned. To solve the Chinese remainder problem

z == xi mod yi for i = 1 to n

use Chin1(x1, y1) to enter the first congruence and then use Chin(xi, yi) to enter each of the congruences with i > 1. The answer z is returned and stored on the list as X1. Y1 on the list is set to the new modulus. All z == X1 mod Y1 are solutions, X1 is just the smallest nonnegative one.

For example, an input of

Z = Chin1(2, 3) Chin(3, 5) Chin(2, 7)

will give Z = 23, X1 = 23, and Y1 = 105. So the answers are 23 + 105*n; n= 0, 1, 2, ...

Concat(X, Y) = Concatenate the columns of matrix X and Y:

Returns a matrix with the columns of X followed by the columns of Y.

ConcatR(X, Y) = Concatenate the rows of matrix X and Y:

Returns a matrix with the rows of X followed by the rows of Y.

CuRt(X) = CubeRoot(X), Re = Remainder:

The cube root function. The remainder, Re = x - y^3 where y = CuRt(x), is also computed and added to the list. The sign of y and x are all the same. For x >= 0, 0 <= Re <= 3*y*(y+1). For example CuRt(26) = 2, Re = 18, CuRt(-26) = -3, Re = 1.

Del(M, C) = Delete column C~ of matrix M:

Returns matrix M with the C-th column deleted. If the last column is deleted, the result will be the scalar zero.

DelR(M, R) = Delete row R~ of matrix M:

Returns matrix M with the R-th row deleted. If the last row is deleted, the result will be the scalar zero.

Det(M) = Determinate of matrix M:

Computes the determinant, a scalar, of the n by n square matrix M. The determinant is zero iff the matrix is singular.

Since only large integers are available to the calculator, the determinant is computed by Sum (-1)^k * M1s1*M2s2*...*Mnsn where the indices s1,...,sn form a

permutation of the numbers 1,...,n. This is summed over all unique permutations. There are n! of them.

Diagonal(V) = Diagonal matrix from a column or row vector V:

Returns a diagonal matrix with its diagonal elements equal to the column or row vector V and the off diagonal elements equal to zero.

Dig(X, Y) = Number of base Y digits in X, Y >= 2:

For example, Dig(5, 2) = 3 (5 = 101 base 2). The signs of X and Y are not used. If |Y| < 2, Y is changed to 10 and a message is generated. Y can be larger than 36. If X == 0, the answer is 0. If Y == X, the answer is 2. If Y > X, the answer is 1.

DigD(X) = Number of decimal digits in X:

For example, DigD(427) = 3. If X == 0, the answer is 0.

DivInt(X, Y) = Floor(X/Y), integer divide, element by element:

Integer divide, DivInt(X, Y) = Floor(x/y). An error message is given if y = 0.

DivRem(X, Y) = Floor(X/Y) and set Re to remainder, element by element:

Integer divide, q =DivInt(X, Y) = Floor(x/y). The remainder is also computed and added to the list as an item names Re. An error message is given if y = 0. x = q * y + Re. The remainder always has the same sign as y or equal to 0, 0 <= |Re| < |y|. If Re is on top of the list, an error message is displayed: " Cannot set same location to both quotient and remainder, continuing...".

Euler(X) = Euler number E(X):

This function returns the exact integer value of the Euler number E(x). If x < 0, 0 is returned. If x is too large, 0 is returned and an error message is generated.

EulerN(X) = Numerator of E(X) = E(X):

Same as Euler(X).

Extract(M, C) = Extract column C~ from matrix M:

Returns a column vector equal to the C-th column of M.

ExtractR(M, R) = Extract row R~ from matrix M:

Returns a row vector equal to the R-th row of M.

Fac(X) = Factorial of X by binary splitting:

Factorial function = 1 * 2 * 3 * ... * x. Error if x < 0. See F or ! => Factorial by binary splitting above

Fac2(X, Y) = Fac(X) / Fac(Y) by binary splitting:

Uses the same recursive algorithm that factorials use, except Fac(n) = Pr(n, 0) and Fac2(a, b) = Pr(a, b).

FacM(X) = (Factorial of X) Mod FMB by binary splitting:

The factorial function with modulo arithmetic. The modulo process is performed after each multiply to prevent the intermediate results from becoming large. If FMB is not on the list, it is added to the list with a value of zero. If FMB is zero, the modulo is not performed. If FMB != 0, the answer a is such that 0 <= |a| < |FMB| and Sign(a) = Sign(FMB) or a = 0.

FacM2(X, Y) = Fac(X) / Fac(Y) Mod FMB by binary splitting:

Same as Fac2(X, Y), but with modulo arithmetic like FacM(X).

FacMS(X) = (Factorial of X) Mod FMB by standard method:

Same as FacM(X) except does not use binary splitting.

FacS(X) = Factorial of X by standard method:

Same as Fac(X) except does not use binary splitting.

Fib(X) = FibonacciNumber(X), Fib(0) = 0:

Fib(N) returns the N'th Fibonacci number. Fib(0) = 0, Fib(1) = 1, Fib(n+2) = Fib(n) + Fib(n+1) for all n. If n is odd, Fib(-n) = Fib(n). If n is even, Fib(-n) = -Fib(n).

Gam(X) = GammaFunction(X) = (X-1)! by binary splitting:

Gam(x) = 1 * 2 * 3 * ... * (x-1). Error if x <= 0.

GCD(X, Y) = Greatest Common Divisor:

Greatest common divisor function. Uses the oldest algorithm in the book, Euclid's algorithm (see Euclid's Elements, Book 7, Propositions 1 and 2). The signs of x and y are not used in the computation because gcd(x, y) = gcd(|x|, |y|). For example, the GCD of 12 and -18 is 6.

GCDe(X, Y) = Extended GCD(X, Y) = X*X1 + Y*Y1:

The extended GCD returns g = GCD(x, y) >= 0 and also computes x1 and y1 such that x*x1 + y*y1 = g. This is useful for computing modular multiplicative inverses. The signs of x and y are used in the computation. For example, the GCDe of 12 and -18 is 6 with x1 = -1 and y1 = -1 since -1*12 + -1*-18 = 6. The GCDe(-99, 78) = 3 with x1 = 11 and y1 = 14 since 11*-99 + 14*78 = 3. The named numbers “X1” = x1 and “Y1” = y1 are added to the named number list.

Get(M, R, C) = Returns element at row R~, column C~ of matrix M:

R and C are scalars. M is a matrix with at least R rows and C columns.

Insert(M, V, C) = Insert column vector V at column C~ of matrix M:

Return matrix M with column vector V at column C.

InsertR(M, V, R) = Insert row vector V at row R~ of matrix M:

Return matrix M with row vector V at row R.

Inv(X, Y) = Z = Inverse of X Mod Y, X*Z == 1 Mod Y:

Returns z (0 < z < y) such that x*z == 1 Mod y or 0 (False) if no inverse exists, i.e. if x and y are not relatively prime. The sign of y is not used.

IsDiag(M) = Returns 1 if M is a diagonal matrix, else 0:

If M is a square matrix with all off diagonal elements equal to zero, the result is 1, else 0. If M is a scalar, 0 is returned.

IsSq(X) = 1 (True) if X is a square, else 0:

Set equal to 1 (True) if x is a perfect square, else it is set to 0 (False). This is computed by factoring x and examining the powers of the prime factors. x is a square iff all of the powers are even. The first prime found with an odd power causes the factoring to stop, and an answer of 0 (False) is given. It may be faster to use the SqRt(x) function and check the remainder in Re for zero if the number is hard to factor.

Kron(X, Y) = Kronecker-Legendre symbol X over Y:

The Kronecker symbol is an extension of the Jacobi symbol to all integers. It is variously written as (x/y) or (x|y) or (x vinculum y). Some values are:

y =:	-9	-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9
x = -9:	0	-	+	0	-	-	0	-	-	0	+	+	0	+	+	0	-	+	0
x = -8:	-	0	+	0	+	0	-	0	-	0	+	0	+	0	-	0	-	0	+
x = -7:	-	-	0	+	+	-	+	-	-	0	+	+	-	+	-	-	0	+	+
x = -6:	0	0	-	0	-	0	0	0	-	0	+	0	0	0	+	0	+	0	0
x = -5:	-	+	-	+	0	-	-	+	-	0	+	-	+	+	0	-	+	-	+
x = -4:	-	0	+	0	-	0	+	0	-	0	+	0	-	0	+	0	-	0	+
x = -3:	0	+	-	0	+	-	0	+	-	0	+	-	0	+	-	0	+	-	0
x = -2:	-	0	+	0	+	0	-	0	-	0	+	0	+	0	-	0	-	0	+
x = -1:	-	-	+	+	-	-	+	-	-	+	+	+	-	+	+	-	-	+	+
x = 0:	0	0	0	0	0	0	0	0	+	0	+	0	0	0	0	0	0	0	0
x = 1:	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
x = 2:	+	0	+	0	-	0	-	0	+	0	+	0	-	0	-	0	+	0	+
x = 3:	0	-	-	0	-	+	0	-	+	0	+	-	0	+	-	0	-	-	0
x = 4:	+	0	+	0	+	0	+	0	+	0	+	0	+	0	+	0	+	0	+
x = 5:	+	-	-	+	0	+	-	-	+	0	+	-	-	+	0	+	-	-	+
x = 6:	0	0	-	0	+	0	0	0	+	0	+	0	0	0	+	0	-	0	0
x = 7:	+	+	0	+	-	+	+	+	+	0	+	+	+	+	-	+	0	+	+
x = 8:	+	0	+	0	-	0	-	0	+	0	+	0	-	0	-	0	+	0	+
x = 9:	0	+	+	0	+	+	0	+	+	0	+	+	0	+	+	0	+	+	0

where + is for +1 and – is for -1. For example Kron(3, 7) = -1, Kron(7, 3) = +1. This function is used internally by the Adleman function for prime number testing, and is part of the prime factorization by Elliptic Curve Method (ECM).

LCM(X, Y) = Least common multiple X, Y:

Least common multiple function. LCM(x, y) = x * y / GCD(x, y). Only the absolute values of x and y are used in the computation. The LCM of 12 and -18 is 12 * 18 / 6 = 36. Special case LCM(0, 0) = 0.

Mag(X, Y) = SqRt(Sq(X), Sq(Y)), Re = Remainder:

Sets A to the integer squareroot of X*X + Y*Y and set the item Re equal to the remainder of the squareroot.

Mat(R, C) = Return a zero matrix with R~ rows and C~ columns:

R and C are scalars. All R * C elements of the returned matrix are set to zero. If R or C < 1, the scalar zero is returned.

MatId(X) = Equals an X~ by X~ identity matrix:

Returns an identity matrix with X columns and X rows. X and must be greater than zero.

Max(X, Y) = Greater of X and Y:

If x >= y, the result is x, else the result is y.

MEq(X) = Mersenne equation = 2^X – 1:

If X is prime, MEq(X) is a Mersenne number. If MEq(X) is prime, MEq(X) is a Mersenne prime. See http://www.geocities.com/hjsmithh/Perfect/Mersenne.html .

Min(X, Y) = Lesser of X and Y:

If x >= y, the result is y, else the result is x.

Mod(X, Y) = X Modulo Y:

Modulo function. r = Mod(x, y) = x Mod y , 0 <= |r| < |y|, Sign(r) = Sign(y) or r = 0, and x = q * y + r, and q the integer quotient of x / y. An error message is generated if y = 0: Cannot divide by zero, continuing... .

Mord(A, N) = Multiplicative order of base A (mod N) or 0:

The multiplicative order function Mord(a, n) is the minimum positive integer e for which a^e == 1 mod n, or zero if no e exists. If a or n is less than 2, zero is returned.

e = Mord(a, n) // e = Multiplicative order
// multiplicative order of base a (mod n) or zero if it does not exist. From Henri Cohen's book,

```
// A Course in Computational Algebraic Number Theory, Springer, 1996, Algorithm
1.4.3, page 25.
{
  e = 0
  if (a <= 1 or n <= 1) return;
  if (GCD(a, n) != 1) return;
  h = Phi(n); // Euler's Totient Function
  factor h by ECM (Elliptic Curve Method)
  // h = Prod(pi^vi) i = 1, ..., k
  e = h;
  for (int i = 1; i <= k; i++)
  {
    e = e / pi^vi;
    g1 = a^e mod n
    while (g1 != 1)
    {
      g1 = g1^pi mod n
      e = e * pi
    }
  }
}
```

MPG(X) = The X'th Mersenne Prime Generator, MPG(1) = 3:

If 1 <= X <= 46, m = MPG(X) is the X'th Mersenne prime.

MPP(X) = Mersenne Prime Power, MPP(1) = 2:

If 1 <= X <= 46, p = MPP(X) is the power that makes 2^p - 1 the X'th Mersenne prime. MPP(1) = 2, MPP(46) = 43112609. MEq(MPP(X) is the X'th Mersenne prime. PEq(MPP(X) is the X'th perfect number.

MPrime(X) = 1 (True) if 2^X - 1 is a Mersenne Prime else 0:

This uses the Lucas-Lehmer-Test to determine if 2^x - 1 is a Mersenne Prime.

```
boolean b = MPrime(x)  // this is the Lucas-Lehmer-Test
{
  if (x == 2) return true;
  if (x < 2 || x is not prime) return false;
  m = 2^x - 1;
  u = 4;
  for (i = 3; i <= x; i++)
  {
    u = (u*u - 2) mod m;
  }
  if (u == 0)
    return true;
  else
    return false
}
```

Mu(X) = Moebius Mu(X) function:

The Moebius or Möbius Mu(n) function is a number theoretic function defined by:

Mu(n) = 0 if n has one or more repeated prime factors
Mu(n) = 1 if n = 1
Mu(n) = (-1)^k if n is the product of k distinct primes

Mu(n) = Mu(-n) (Mu(n) is not normally defined for n < 1)
Mu(0) = 0

so Mu(n) = 0 iff n is not square-free. If n is square-free, the sign of Mu(n) tell whether there is even or odd number of distinct prime factors (+1 for even, -1 for odd). For n = 0, 1, 2, ... the first few values are 0, 1, -1, -1, 0, -1, 1, -1, 0, 0, 1, -1, 0, ...

P(X) = The X'th prime:

P(n) is the n'th prime. P(1) = 2, P(2) = 3, ... P(0) is defined to be 0. This is the inverse of the function Pi(x). In other words, P(n) is the smallest x for which Pi(x) = n.

Pascal(X) = Equals an X~ by X~ Pascal triangle Pij = Bino(i,j):

The upper triangle above the diagonal will be all zeros.

PEq(X) = Perfect equation = (2^X - 1) * 2^(X-1):

If MEq(X) is prime, MEq(X) is a Mersenne prime and PEq(X) is an even perfect number. All even perfect numbers are of this form. See http://www.geocities.com/hjsmithh/Perfect/Mersenne.html .

PGT(X) = First prime > X:

What more can I say. PGT(0) = PGT(1) = 2, PGT(2) = 3, PGT(3) = 5, ... If x < 0 its sign is ignored. PGT(-Abs(x)) = PGT(Abs(x)). The Adleman function is used to determine if a number is prime.

Phi(X) = Euler's totient function:

Phi, Sig (sigma) and Tau are number theoretic functions. Phi(n) = number of positive integers not exceeding n and relatively prime to n. For example, Phi(60) = 16. The 16 integers are 1, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 49, 53, and 59. Phi(0) is defined to be 0. Phi(1) = 1. Phi(2) = 1. Phi(3) = 2.

The classic formula for Phi is:

Phi(n) = n * (1-1/P1) * ... * (1-1/Pr)

but for computation the following formula is used

Phi(n) = ((P1-1)*P1^(A1-1)* ... * (Pr-1)*Pr^(Ar-1)

For this document the following notation is used. An integral number n > 1 can be written as n = P1^A1 * P2^A2 * ... Pr^Ar where the Pi's are the various different prime factors, Ai the number of times Pi occurs in the prime factorization and r the number of prime factors.

PhiL(X, A) = Legendre's formula:

PhiL(x, a) = Legendre's formula i.e., the number of integers in [1, x] not divisible by any of the first a primes. This function is used by the prime counting functions Pi(x), PiL(x), PiL1(x), and PiM(x). For values of A larger

than about 13000, this function is prone to a Run-time error of "System.StackOverflowException was thrown", but this will not crash the program.

Pi(X) = Number of primes <= X by sieve or Lehmer:

This is the prime counting function, for every real x >= 0, Pi(x) is the number of primes p such that p <= x. For example Pi(5) = Pi(6) = 3. The 3 primes are 2, 3, and 5. For small x, the sieve of Eratosthenes is used, for large x, Lehmer's formula is used. This is the method to use unless you are testing PiL1(x) (slowest), PiM(x) (slow), or PiL(x) (fastest for large x).

PiL(X) = number of primes <= X by Lehmer's formula:

This is the prime counting function using Lehmer's formula. This is the fastest of the three functions PiL1(x), PiM(x), and PiL(x).

PiL1(X) = number of primes <= X by Legendre's formula:

This is the prime counting function using Legendre's formula.

PiM(X) = number of primes <= X by Meissel's formula:

This is the prime counting function using Meissel's formula.

PLT(X) = Largest prime < X:

Finds the next prime P < x. By definition, PLT(0) = PLT(1) = PLT(2) = 0 (False). The sign of the input x is ignored. The running time of this and the other commands that requite prime factoring can be quite long if a number having large factors is encountered. The Adleman function is used to determine if a number is prime.

PNG(X) = The X'th Perfect Number Generator, PNG(1) = 6:

If 1 <= X <= 46, p = PNG(X) is the X'th perfect number.

Pow(X, Y) = X to the power Y:

The Exponential function. The peasants' method is used in which up to 2 * Log base 2 of y multiplies of powers of x are done to compute the result. If x = 0 and y = 0, an answer of 1 will be given. An error message is generated if x = 0 and y is < 0: Cannot raise zero to a negative power.

I have speeded up x=a^b when a=10 or a=-10 and b >= 0, especially for large precision. Also works for Pow(a, b) but not PowM(a, b).

PowM(X, Y) = (X to the power Y) Mod FMB:

The Exponential function with modulo arithmetic. The peasants' method is used in which up to 2 * Log base 2 of y multiplies of powers of x are done to compute the result. The modulo process is performed after each multiply to prevent the intermediate results from becoming large. If FMB is not on the list, it is added to the list with a value of zero. If FMB is zero, the modulo is not performed. If x = 0 and y = 0, an answer of 1 will be given. An error message is generated

if x = 0 and y is < 0: Cannot raise zero to a negative power. The answer z is such that 0 <= |z| < |FMB| and Sign(a) = Sign(FMB) or z = 0.

Prime(X) = 1 (True) if X is Prime else 0:

If x is a prime number then Prime(x) = 1 (True). If x is not prime, Prime(x) = 0 (False).

Primo(X) = Primorial function, product of all primes <= X:

The primorial function is a cross between the prime counting function and the factorial function. For every real x >= 0, Primo(x) is the product of all primes p such that p <= x. For example Primo(5) = Primo(6) = 2*3*5 = 30. Primo(0) = Primo(1) = 1. The primorial function is undefined for x < 0.

PrimR(X) = Largest and smallest primitive root of X or 0:

The number of primitive roots (pr's) that x has is displayed. If x does not have any pr's, zero is returned. If x has only one pr, it is returned. If x has 2 or more pr's, the largest and smallest are displayed and the smallest is returned. The named number "Re" is set to the largest primitive root or 0 if x does not have any pr's. For example:

```
Command: X = PrimR(41)

The prime number 41
has 16 primitive roots, the largest is 35 = n - 6, the smallest is 6

X = 6

Command: Re=

Re = 35

Command: X = PrimR(137842

The non-prime number 1,37842 = 2^1 * 41^3
has 26240 primitive roots, the largest is 1,37835 = n - 7, the smallest is 7

X = 7

Command: Re=

Re = 1,37835
```

PrimRP(X) = Largest and smallest prime primitive root of X:

The number of primitive roots (pr's) that x has is displayed. If x does not have any prime pr's, zero is returned. If x has only one prime pr, it is returned. If x has 2 or more prime pr's, the largest and smallest prime pr are displayed and the smallest is returned. The named number "Re" is set to the largest prime primitive root or 0 if x does not have any prime pr's.

PrimRQ(X, Y) = 1 (True) if X is a primitive root of Y, else 0:

This function answers the question: Is x a primitive root of |y|?

Rev(X) = Digit Reversal of X base 10:

The decimal digits of x are reversed MSD to LSD. Rev(1234) = 4321.

Rev(X, Y) = Digit Reversal of X base Y:

The base-y digits of x are reversed MSD to LSD. Rev(1234, 5) = 2746 because 1234 = 14414 (base 5) and 41441 (base 5) = 2746. If y < 2, base 10 is used. Rev(X, 0) = Rev(X, 10) = Rev(X).

RInt(X, Y) = RandomInteger between X and Y inclusive:

Generates a random integer z, x <= z <= y or y <= z <= x if x > y. Repeat this function, F4, to generate a sequence of random integers. You should not use RN in the command, like RN=RInt(RN, RN).

RN(X) = RandomNumber(X=Seed):

Random number function. RN(x) evaluates to a random number between zero and 10^35. This number will never have more than 35 decimal digits. Theoretically the random number generator will cycle after 10^35 numbers, but the earth will not last that long. The argument of the function is taken as the seed of the random number generator. For a consecutive set of random numbers, the argument x should be the previous random number generated. The items RNA, RNC, and RN are put on the list by the random number function. The equation used is: RN(x) = (RNA * x + RNC) mod (10^35), where RNA and RNC are 35 digit integers. RN is put on the list so it will be used as the seed the next time the N command is used.

```
RNA = 184362483057250753463742919207653 41,
RNC = 863464797320479456723825446396254 43.
```

SE(X) = X Squared, element by element:

The square function, x @ x, element by element.

Set(M, R, C, Y) = Return M with element at row R~, col. C~ set to Y~:

R, C and Y are scalars. M is a matrix with at least R rows and C columns. Z = Set(M, R, C, Y) will not change M.

SetAll(M, Y) = Return matrix M with all elements set to Y~:

Y is a scalar and M is a matrix. Z = SetAll(M, Y) will not change M.

Sig(X) = Sum of divisors of X:

Sig(n) = sum of the positive integral divisors of n. Sig is short for sigma the 18'th letter of the Greek alphabet. Sig(0) = 0, Sig(1) = 1, Sig(2) = 3.

The formula for this is:

```
Sig(n) = ((P1^(A1+1) - 1) / (P1-1)) * ... * ((Pr^(Ar+1) - 1) / (Pr-1))
```

Sig0(X) = Sum of divisors of X (-X):

Sig0(n) = sum of the positive integral divisors of n, not including n. Sig0(n) = Sig(n) - n. Sig(n) is the classical number theoretic function and Sig0 is handy for check perfect numbers. A number n is a perfect number iff n > 0 and Sig0(n) = n. Sig0(0) = Sig0(1) = 0, Sig0(2) = 1, ..., Sig0(6) = 6. Six is the first perfect number.

Sign(X) = 0 if X=0, else = X / |X|:

Sign(x) = -1 if x < 0, = 0 if x = 0, = +1 if x > 0.

Size(X) = {r, c} where r = rows in X, c = columns in X:

Size(X) is a two element row vector {r, c} with r = the number of rows in X and c = the number of columns in X. If X is a scalar, the result is zero.

Solve(X, Y, N) = Solve for z, X * z == Y Mod N:

Returns z equal to the minimum solution of the modular linear equation x * z == y Mod n or zero if no solution exists. A solution exists iff GCD(x, n) is a divisor of y. If there is more than one solution, all solutions are displayed. The absolute value of N is used. For example,

Command: z = solve(35, 10, 20)

Solution 5: 14
Solution 4: 10
Solution 3: 6
Solution 2: 2
Solution 1: 18

z = 2

Command: z = solve(35, 1, 20

Solution: No solution

z = 0 (False)

Sord(A, N) = Multiplicative suborder of base A (mod N) or 0:

The multiplicative suborder function Sord(a, n) is the minimum positive integer e for which a^e == +/-1 mod n, or zero if no e exists. If A or N is less than 2, zero is returned.

```
e = Sord(a, n) // e = Multiplicative Suborder
// multiplicative suborder of base a (mod n) or zero if it does not exist.
// From Stephen Wolfram, Algebraic Properties of Cellular Automata (1984).
// Sord = Mord or Mord/2, Mord/2 iff a^(Mord/2) mod n = -1
{
  e = Mord(a, n)
  if ((e is odd) or e == 0), return
  e = e/2
  g1 = a^e mod n
  if (g1 != (n-1)) e = 2*e
}
```

SortC(X) = Sort each column of matrix X in numerical order:

If x is a scalar or a row vector, it is left unchanged.

SortCS(X, Y, Z) = Sort each column of matrix X, row Y thru Z:

Same as SortC(X) except only rows Y through Z are included in the sort. If Y is a matrix or less than 1, it considered equal to 1. If Y is larger than the number of rows in X, it is considered equal to the number of rows. If Z is a matrix or larger than the number of rows in X, it considered equal to the number of rows. If Z is less than 1, it is considered equal to 1. If Z < Y, rows Z through Y are in the sort.

SortR(X) = Sort each row of matrix X in numerical order:

If x is a scalar or a column vector, it is left unchanged.

SortRS(X, Y, Z) = Sort each row of matrix X, column Y thru Z:

Same as SortR(X) except only column s Y through Z are included in the sort. If Y is a matrix or less than 1, it considered equal to 1. If Y is larger than the number of columns in X, it is considered equal to the number of columns. If Z is a matrix or larger than the number of columns in X, it considered equal to the number of columns. If Z is less than 1, it is considered equal to 1. If Z < Y, columns Z through Y are in the sort.

Sq(X) = X Squared, matrix multiply:

The square function, x * x, matrix multiply.

SqFree(X) = 1 (True) if X is a squarefree, else 0:

Set equal to 1 (True) if x is squarefree, else it is set to 0 (False). This is computed by factoring x and examining the powers of the prime factors. x is squarefree iff all of the powers are one. The first prime found with a power > 1 causes the factoring to stop, and an answer of 0 (False) is given.

SqRt(X) = SquareRoot(X), Re = Remainder:

The positive square root function, error if x < 0. The remainder, Re = x - Sq(SqRt(x)), is also computed and added to the list. The remainder is always >= 0, 0 <= Re <= 2*SqRt(x). For example SqRt(24) = 4, Re = 8.

SumD(X, Y) = Sum of base Y digits in X, Y >= 2:

For example, SumD(5, 2) = 1 + 0 + 1 = 2 (5 = 101 base 2). The signs of X and Y are not used. If |Y| < 2, Y is changed to 10 and a message is generated. Y can be larger than 36. If Y == X, the answer is 1. If Y > X, the answer is X.

SumDD(X) = Sum of decimal digits in X:

For example, SumDD(427) = 4 + 2 + 7 = 13.

Tau(X) = Number of divisors of X:

Tau(n) = number of positive integral divisors of n. Tau(0) = 0, Tau(1) = 1. The formula for Tau is:

Tau(n) = (A1+1) * ... * (Ak+1)

When the command AllD(X), IsSq(X), Mu(X), PFA(X), PFE, Phi(X), Sig(X), Sig0(X), SqFree(X), or Tau(X) completes the factorization of x, this factorization is saved. When one of these commands is executed with x = 0 and there is a saved factorization, the saved factors and exponents are used to complete the command. This can save a lot of time when more than one of these commands are needed for the same number. To see the saved factorization, try PFE(0).

Executing one of these commands with x = 1 will delete the saved factorization. The commands Prime(X), PFB(X), PGT(X), and PLT(X) will also delete any saved factorization. The prime related commands Pi(X), P(X), and PTab(X) have no effect on the saved factorization.

Trace(M) = Sum of elements on the principal diagonal of M:

Trace(M) = m11 + m22 + m33 + ... M must be a matrix but it does not need to be square.

Tran(M) = Transpose of matrix M”

If M is an r by c matrix, the result will be a c by r matrix with the j, i element equal to the i, j element of M.

{M11, M12, ...; ..., Mrc} = Enter a matrix with r rows and c columns, Mij~:

For example, {1, 2; 3, 4; 5, 6} will enter the matrix

1 2
3 4
5 6

The closing } is not required iff it would be the last character on the command line. This function can be used in expressions like any other function. The first row sets the column size. If some rows are shorter than this, they are entered as zero.

-X = Negative of X, 0 - X:

Negative inverse of x. The -, +, and ! functions do not require the parentheses so they also can be considered as unary or monadic operators.

+X = Positive of X, 0 + X:

The identity operator, +x = x.

!X = Not X, 0 -> 1 else 0:

Logical Not operator. Not !X (!!X) will leave 0 alone and will change all other values to 1 (True).

Function keys -

The function keys described here or used on the Run form.

F1 => Display Help form, same as the "?" primitive.

F2 => Totally Quit/end the program, same as the "Q" primitive:

If a calculation is running, indicated by the word **Running** displayed on the Run form, when F2 is pressed, it is treated as an Abort Calc. request.

F3 => Restore previous input command:

The F3 key normally will restore the command input text box to the value of the previously executed command input. After the B, K, or X command is executed, this key will restore the command line with the learned line. If the learned line changes, when it executes, the previous value of the learned line will be restored by this key.

F4 => Restore previous input command and accept:

The F4 key is the same as F3 except that the previous command is executed without the Enter key being required.

F5 => Get Status of calculation:

Press the F5 key while a computation is being performed and a message like "Integer Multiply: 50 Percent done" will be displayed in the output text box. This is the same as the "Status of Calc." command button.

F6 => Display Configuration form:

Press the F6 key and the Configuration form will appear. F6 is the same as the "Configuration" command button.

F7 => Display Restore Input History form:

Press the F7 key and the Restore Input History form will appear. F7 is the same as the "Restore Input" command button.

F8 => Accept input and Calculate:

This will cause a non-empty contents of the input text box on the Run form to be accepted as command input. Pressing the enter/return key when the cursor is at the end of the command input will accomplish the same thing. F8 is the same as the "Calculate" command button.

F9 => Toggle Logging to Log file on/off:

If the "Logging screen to log file mode is off it will be turned on, if on it will be turned off. F9 is the same as the "Logging is On/Off" command button and the same as the "H" primitive.

F11 => Clear output text box:

This clears the output text box. F11 is the same as the “Clear Output” command button.

F12 => Pause:

This causes the program to suspend all operations until the operator clicks the “OK” button to resume. This frees up the computer if the processor is needed for a priority task. The timing function provided by the Diag command is also suspended so it will continue to give good timing information. F12 is the same as the “Pause” command button and the XJCalc Pause command

Ctrl+F2 => Restart ($).

Ctrl+F9 => Clear Log File (ClearLog).

Ctrl+F11 => Clear input text box:

This clears the input text box so you can start typing a new command. The box is also cleared when a command is accepted for execution, so this key is not needed in that case. Ctrl+F11 is the same as the “Clear Input” command button. In Windows the F10 key is used to activate the file menu button in the upper left hand corner of the form so F10 is no longer used in this program.

Ctrl+S => Save All (Save).

Ctrl+O => Restore All (Restore).

ESC => Clear Run form Input Text Box:

If a calculation is not currently running, the escape key on the RUN form clears the input text box. On the Startup, Help, About, Restore Input History, Configuration, and Change Disk Directory form, the escape key exits the form.

ESC => Interrupt/Abort a long process:

The ESC key from the Run form while a calculation is running is the same as the “Abort Calc” command button. It can be used after the program has been asked to perform a task that is taking longer than the operator is willing to wait. Press the ESC key once and the message box with the message:

```
*** INTERRUPT: INTERRUPT: To Continue select Ignore
To Abort Computation select Abort
To Set SoftAbort flag select Retry
```

will appear.

If Ignore is selected, the interrupt is ignored.

If Abort is selected or Space bar, Enter, or the “A” key is pressed, the message:

```
Computation aborted by operator!
```

will appear, and if auto display is on, the value of the currently active item will be displayed. If the ESC key is pressed during the display of a value, the

same messages will appear, but if Abort is selected, it is still displayed correctly but with fewer digits. Pressing ESC and aborting during execution of the Z command causes all item on the list not yet displayed to be displayed with fewer digits.

If Retry is selected, the message:

    SoftAbort flag set by operator!

will appear. When the SoftAbort flag is set, the variable SoftAbort is put on the list and is set to a value of 1. Its purpose is to allow the operator to flag a XJCalc Code file that it should gracefully terminate its operation. This flag has no other purpose. The SoftAbort item remains =1 until a code file or the operator changes it.

There is another variable that the calculator can add to the list. The variable muErr is put on the list and is set to a value of 1 (True) when a multi-precision error occurs. Its purpose is to allow a code file to detect when it has caused an error and adjust or terminate.

Page Up key => Display previous newer command:

The page up and page down keys can be used on the Run form to move newer and older previous commands into the input text box for execution.

Page Down key => Display previous older command:

The page up and page down keys can be used on the Run form to move newer and older previous commands into the input text box for execution.

Commands used in XJCalc code files:

The following commands are primarily for use in XJCalc code files, but can be used from the operator command input text box.

If Command:

The If command is the first word of an If statement. The syntax of the If statement is:

    If {expression} Then {statements} Else {statements}

The expression following the If is evaluated and if it is True, i.e., not zero, all statements on the same line following the next Else are deleted and execution continues with the statements following the Then. If the expression evaluates to zero (False), all statements following the expression up to the next Else are deleted and execution continues with the statements following the next Else. The Then key word is optional, the Then {statements} is optional and the Else {statements} is optional.
The equivalent of a case statement can be constructed for example like:

```
If A=1 B=3 Else If A=2 B=5 Else If A=3 B=7 Else B=0
```
This is equivalent to:

```
B=0 If (1 <= A) & (A <= 3) Then B=2*A+1
```

GoTo Command:

The GoTo {label} command will skip all statements following the GoTo until {label}: is found and then start executing the statements following the {label}:. If the GoTo command is in a XJCalc code file, lines of input also will be skipped until the {label}: is found or until an end-of-file. It is not an error if the {label}: is not found, but a GoTo end-of-file or end-of-line will be performed in this case.

GoUpTo Command:

The GoUpTo {label} command will skip all statements following the start of the current line until {label}: is found and then start executing the statements following the {label}:. If the {label}: is not found on the current line and the GoUpTo command is in a XJCalc code file, the file will be reset to the first line of the file and lines of input will be skipped until the {label}: is found or until an end-of-file. It is not an error if the {label}: is not found, but a GoTo end-of-file or end-of-line will be performed in this case.

Labels:

A label is a name followed by a colon (:). When encountered as a command, a label is a no-op. When searching for where to go from a GoTo {label} or from a GoUpTo {label} command, the {label}: is used to determine where to restart execution. If duplicate labels are on a command line or in a code file, the first one encountered is the one that is effective. Labels are alphanumeric with the first character alphabetic, have all characters significant but not case sensitive. Other non-delimiter characters can be used in labels, but this is not recommended. The delimiter characters are:

, ; < = > + - ! | * / & : ( ) ^ @ # % \ ' "

Continuation lines:

Continuation lines are indicated by the last non-blank character of the line being a + or - character. A + says, this line is to be continued by adding the next line, but a blank character should be included between them if it is needed to separate fields. A - says, this line is to be continued by adding the next line, but no extra blank characters should be included between them.

Batch Commands (Echo, @Echo, Pause, and Rem) -

Echo Command:

Normally, commands from a XJCalc code file are displayed on the screen as they are executed. This can be turned off by the Echo off command and turned on by the Echo on command. If something other than or more than on or off follow the word Echo, it is considered a message and is output to the screen.

@Echo Command:

The @Echo command is the same as the Echo command except that, if it is the first command on a line, it is executed before the command line is echoed to the screen. Thus, an @Echo off at the beginning of a line will do an Echo off without the command being echoed to the screen.

Pause Command:

The pause command will output the following message to a Message Box and wait for operator to resume:

XJCalc is Paused. OK to resume?

The escape key, space bar, enter/return key, or selecting “OK” will clear the pause action.

Rem Command:

The syntax of the Rem command is Rem {remark}. It is a no-op command and everything on the line following the word Rem is skipped.

// Command:

The // command is the same as the Rem command except that it does not need any spaces or delimiters before or after it.

Command buttons -

There are 13 command buttons on the Run form:

Abort Calc.: Same as ESC key.
Calculate: Same as F8 key.
Clear Input: Same as Ctrl+F11 key.
Clear Log File: Clears the log file, same as the ClearLog command and Ctrl+F9.
Clear Output: Same as F11 key.
Configuration: Same as F6 key.
Logging Is On/Off: Same as F9 key.
Pause: Same as F12 key or Pause command.
Quit: Quit the Run form and go back to the Startup form.
Restore All: Restore Configuration, History, & List, same as Restore command and Ctrl+O.
Restore Input: Same as F7 key.
Save All: Save Configuration, History, & List, same as Save command and Ctrl+S.
Status of Calc.: Same as F5 key.

If the Quit button, the File menu Exit, or the form’s Close button is selected when a calculation is running, indicated by **Running** being displayed on the Run form, the calculation is automatically aborted. The message “When **Running**, select Quit twice to quit” is displayed in the text output text box and a second quit request is required to quit.

The Run form’s Close button, Close menu item and Exit menu item will totally end the program without stopping at the Startup form.

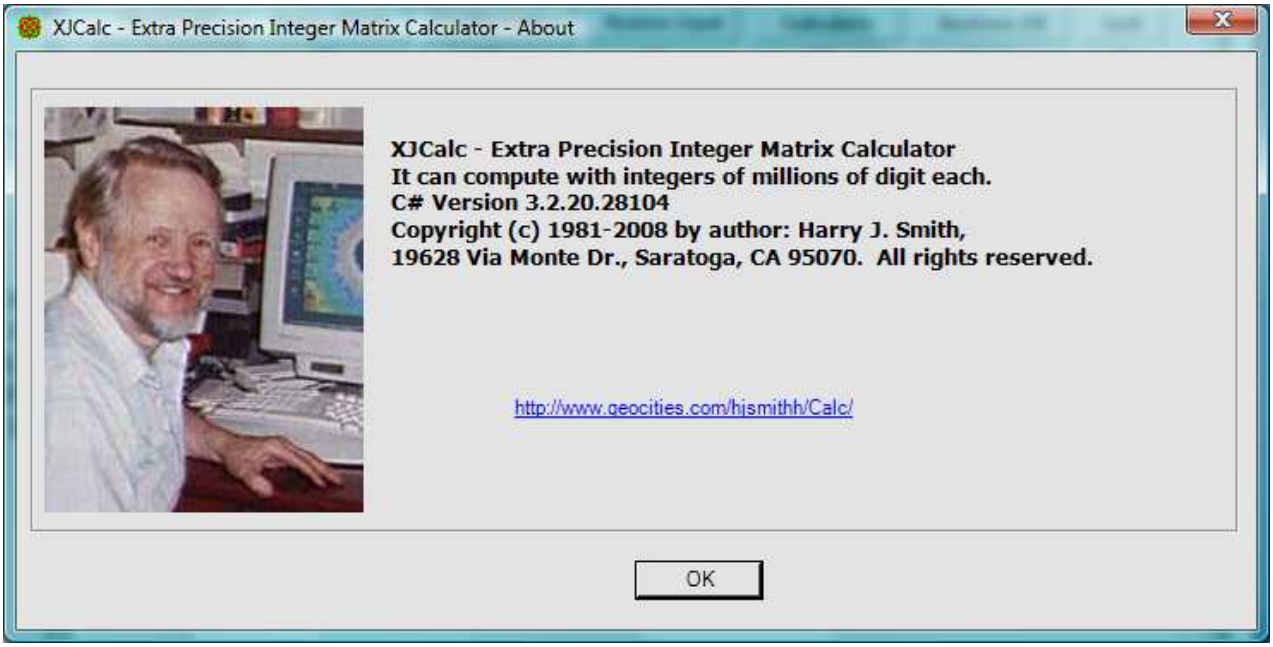
The Run form's Close button, Close menu item and Exit menu item will totally end the program without stopping at the Startup form.

The Run form has a File, a Restore, and a Help menu button. The File menu has Pause, Restart, and Exit. The Restore menu has Restore previous input command and Accept previous input command. The Help menu has Help Form, On Top and About...

Pause is the same as the Pause button.
Restart is the same as the $ command.
Exit is the same as the Quit button.

Restore previous input command is the same as function key F3.
Accept previous input command is the same as function key F4.

Help Form is the same as the ? primitive and causes the Help form to be displayed.
On Top is a check box that, when checked, will keep the Run form on top of other windows.
About will bring up a dialog box with Application Title, Version, Application Description and URL to obtain the latest version of the program:



Click the OK button or hit the escape button to remove this box.

Error reports -

There are many different error reports that are a result of directly or indirectly requesting an operation that cannot be performed (or conceivably an error in the XJCalc program). Another type of error is the syntax error, where a command cannot be interpreted.

The computation error reports are:

```
Addition overflow, continuing... (M=<m>)
Cannot divide by zero, continuing...
Cannot raise zero to a negative power
Cannot set an element of a scalar
Cannot set same location to both cube root and remainder, continuing...
Cannot set same location to both max and min primitive root, continuing..
Cannot set same location to both quotient and remainder, continuing...
Cannot set same location to both square root and remainder, continuing...
Cannot set same location to GCD and extended GCD, continuing...
Cannot set same location to Z and Y1, continuing...
Cannot set Top to FacM(X, Y), FMB cleared
Cannot set Top to PowM(X, Y), FMB cleared
Cannot take factorial of number < zero
Cannot take primorial of number < zero
Cannot take square root of negative number, continuing...
Cannot use a matrix as a scalar
Cannot use a scalar as a matrix
```

```
Element outside of matrix
FHT cannot recover from failure/probable error in FHT multiply
FHT Max Error = {max}, probable error in FHT multiply!!!!!!!
FHT overflow, should not happen but it did!!!!!!!
FHT Recovered from failure/probable error in FHT multiply
FHT Sum-of-Digits failed, error in FHT multiply!!!!!!!
FMB is not a scalar
GenBern: n > 10000000, too large for Bernoulli number
GenBernI: Bernoulli n too large
GenEuler: n > 10000000, too large for Euler number
GenEulerI: Euler n too large
Input number overflow, continuing... (M=<m>)
Matrices are incompatible
Mersenne Prime index out of supported range [1, 46]
Multiplication overflow, continuing... (N=<n> > M=<m>)
Number too large for factorial function
Numbers too large for binomial coefficient
Numbers too large for FHT multiply, n = <n>
One digit multiply overflow, continuing... (M=<m>)
SetTo overflow, continuing... (N=<n> > M=<m>)
Signed Integer shift left overflow, continuing...
This is not a square matrix
Unexpected error in Cube Root function
Unsigned subtraction error, continuing...
X too big to add, continuing... (N=<n> > M=<m>)
```

Some of these error messages are due to internal checks and should never be seen by the user.

The syntax errors are:

```
( expected: {procedure}|{string}
Directory name expected: CHDIR(|{string}
Error in function's 2nd argument: {name}(...,|{string}
Error in function's 3rd argument: {name}(...,|{string}
Error in function's argument: {name}(|{string}
Exponent expected: ^{sign}|{string}
Expression expected: (|{string}
Expression expected: (IF |{string}
Expression expected: {name}(|{string}
Factor expected: {op}|{string}
File name expected: {procedure}(|{string}
Simple Expression expected: {op}|{string}
Term expected: {op}|{string}
Unknown function: |{name}({string}
Unknown operation, Command line discarded: |{string}
```

The vertical bar | always shows the start of the string of characters that cannot be interpreted.

I/O error messages are:

```
Cannot open file “{filename}” for input
Cannot open file “{filename}” for output
File not found
Read error file “{filename}”
```

Other informational messages:

```
(Done {n} of {m}, {g} to go)
```

```
{function name}: 2nd and 3rd argument assumed = 0
{function name}: 3rd argument assumed = 0
{function name}: Only two arguments used
{function name}: Second argument assumed = 0
{function name}: Second argument ignored
{function}: {error message}
{n} Commands read from History file, {m} total
{n} Commands written to History file
Aborting file input...
Aborting file write...
Bernoulli number storage cleared
Command history cleared
Command line was:  {command}
Computation aborted by operator!
Continuing...
CRT: {x}, {y}
Directory changed to {directory}
Directory name = {directory}
Directory not changed {directory}
Euler number storage cleared
Factorial function aborted by operator, {n} multiplies to go
FHT Radix = {b}  ldn = {n}  Max Error = {e}  DT = {t} sec.  x.N = {n} ...
File "{filename}" closed
File "{filename}" opened for reading
File "{filename}" opened for writing
File is corrupted: "{filename}"
Full name = "{directory\filename}"
Generating Bernoulli numbers upto B({x})
Generating Euler numbers upto E({x})
has {n} primitive roots, the largest is {a}, the smallest is {b}
has {n} primitive roots, the largest prime is {a}, the smallest prime is
{b}
Have Bernoulli numbers upto B({n})
Have Euler numbers upto E({n})
Home directory is {directory}
I/O operation aborted by operator!
Inverse: No inverse
Label skipped = {label}:
Log file "{filename}" Cleared {date} {time}
Log file "{filename}" Closed {date} {time}
Log file "{filename}" Opened for Append {date} {time}
Max commands in history changed from {old max} to {new max}
Max commands in history not changed from {max}
MersenneP: u{i} = {n}
Most significant digits discarded in {x}
Normal multiply, baseM = {b}  DT = {t} sec.  x.N = {n}  y.N = ...
Not in current directory {dir}
Not in home directory {dir}
MPrime: (Done {n} of {m}, {g} to go)
OpenAppend: {error message}
OpenRead: {error message}
OpenRead: Could not find file '{directory\filename}'.
OpenWrite: {error message}
Path not found!
Power = {p}
Priority is {priority}
Reading Commands from History file - Begin ... Aborted
Reading Commands from History file - Begin ... End
Reading Help File: "{filename}"
Reading number file "{filename}" No named number found
Reading number file "{filename}" containing {item name}
Rewinding code file
Running code file "{filename}"
```

```
      SoftAbort flag set by operator!
      Solution {n}: {z}
      Solution: No solution
      SortC: No elements were swapped
      SortC: Some elements were swapped
      SortR: No elements were swapped
      SortR: Some elements were swapped
      SumD: Base changed from {b} to 10
      T = x.xx  DT = x.xx sec.  End of execution
      T = x.xx  DT = x.xx sec.  Start execution
      The non-prime number {x} = {factors}
      The non-prime number {x} has 0 prime primitive roots
      The non-prime number {x} has 0 primitive roots
      The non-prime number {x} has 1 prime primitive root equal to {a}
      The non-prime number {x} has 1 primitive root equal to {a}
      The prime number {x} has 0 prime primitive roots
      The prime number {x} has 1 primitive root equal to {a}
      When **Running**, select Quit twice to quit
      Writing all Commands to History file - Begin ... End
      Writing file "{filename}"
      Writing file "{filename}" with {item name}
      ys < qs in SqRtRemFastSI()

Status messages -

      BaseI = {base-in} base ten, Max digit = {x}
      BaseO = {base-out} base ten, Max digit = {x}
      Bino: (Done {n} of {m}, {g} to go)
      Converting number to output base: xx Percent done
      FacMBSI: (Done {n} of {m}, {g} to go)
      FibSI: (Done {n} of {m}, {g} to go)
      Find the least primitive root of x:
      FHT: {xx} Percent done
      FHT0: {xx} Percent done
      GenBernI: (Done {n} of {m}, {g} to go)
      GenEulerI: (Done {n} of {m}, {g} to go)
      Integer Divide: xx Percent done
      Integer Multiply: xx Percent done
      Integer Square Root: xx Percent done
      MuAbort exit GenBern
      MuAbort exit GenEuler
      Went to {label}:

Prime number status and diagnostic messages -

      (Testing {n})
      {a}^{b} mod {m} = {c}
      {d} is a big factor of {n}
      {n} / {d} MethodX
      {n} / {d} MoreDiv
      {n} / {d} SieveMethod", Tot
      {n} ^ 1
      {n} has no small factors
      {n} is a hard nut to crack
      a > 2147483647 too large for Legendre's formula
      Completely factored
      Completely factored (Exponents are in decimal)
      DelX=X-Root(N)={n}
      EC Method with limit (B1 = {b1} B2 = {b2}) Curve 1 2 ...
      ECM: (Done {n} of {m}, {g} to go)
      Enter FactorN
      Enter LargeFactor
      Enter MoreDiv, Try the division method some more
```

```
Enter PowerMod
Enter ReportFactor
Enter SieveMethod
Enter SmallFactors
Error, cannot store more than 2000 prime factors
Error, factors do not check
Exit FactorN
Exit FactorN from LargeFactor
Exit FactorN from ReportFactor
Exit LargeFactor
Exit PowerMod
Exit ReportFactor
Exit SmallFactors
FnAdleman: I quit! Too big.
FnAdleman: Test5_P
Generating prime number table with {n} primes
GetExSI: Specified argument was out of the range of valid values.
I={i} J={j} MATCH={True/False}
In FactorN loop
Is x a prime number?:
IsSq(X) function:
KS({I})={m}
Large factor {f} is prime
Large factor is not prime
LargeFactor exit FactorN
Moebius Mu(X) function:
MoreDiv exit FactorN
MuAbort exit MoreDiv
N is not prime
N is prime
N is too big
N= {n} Start= {s}
Normal exit MoreDiv
{Number} already factored with {n} prime(s)
Number not completely factored
Numbers got too large for prime number generator
P({n}): P({i}) = {p}
P(n) = The n'th prime:
PhiLEx: n = {n}, a = {a}
PhiLInt: n = {n}, a = {a}
PhiLSI: n = {n}, a = {a}, depth = {d}
Prime factor algorithm A:
Prime factor algorithm B:
Prime Greater Than:
Prime Less Than:
Prime Pi Function:
Primo({n}): P({i}) = {p}
PrimePiL: (Done {n} of {m}, {g} to go)
PrimePiM: (Done {n} of {m}, {g} to go)
Prime Phi function = Number of integers <= n relatively prime to x:
PrimRQ function, is x a primitive root of y?:
Prime Sig function = Sum of the divisors of x:
Prime Sig0 function = Sum of the divisors of x (-x):
Prime Tau function = Number of divisors of x:
Primorial Function:
Quick exit SmallFactors
Report={True/False} NF={nf} Po={po} Factor={f}
ReportFactor exit FactorN
ReportFactor exit MoreDiv
ReportFactor exit SieveMethod
ReportFactor exit SmallFactors
Should exit LargeFactor
SieveMethod exit FactorN
```

```
SmallFactors exit FactorN
SqFree(X) function:
Square Max P = {n}
SqRt={x} Rem={r}
Starting to factor
Stored Number not squarefree
Wait, filling the {n} entry prime table
Y=Root(X*X - N)={root} Rem={rem}
```

The (Done {n} of {m}, {g} to go) message can be displayed when the Abort Calc. button is selected. It helps you decide if you really want to abort. {n} is the number of iterations of an innermost loop. {g} is the number of iterations left to go. For example, for the factorial function, {n} is the number of multiplications done and {g} is the number left to do.

Other forms that can be displayed to help in using the program are the Startup, Help, Restore Input History, Configuration, and the Change Disk Directory forms. On all forms except the Run form, pressing the ESC key will remove the form.

Startup form -

The Startup form is shown above and has two parameters that can be set before the calculator starts accepting commands:

“Max Digits in a Number” can be set to a value from 1 to 134218400. XJCalc will change this to the next multiple of 8 greater than or equal to 56. The M and SetMax(X) commands will not be able to set Max digits in a number greater than this.

“Max Commands in History” can be set to a value from 1 to 100000. XJCalc will allocate an array of this length to save command history. The nominal starting value is 1000 and it can be changed by the SetC(X) command

These two values are tested as they are edited. Spaces on the left or right are removed so you will not see them. These numbers are displayed in **bold** face font iff they are acceptable values. If either value is unacceptable, the Run button is disabled.

A “Default Settings” button is provided to reset the two parameters to their original values.

The Run button is used to bring up the Run form. It can be used after a Run form exits to reinitialize XJCalc and start running again. The Exit button will terminate the program.

Help form -

The Help form was shown and partially described above. It has a File menu button that has a menu with Print Setup, Print..., Print Preview and Exit. The Print Setup will bring up the Windows printer setup dialog box so you can select the printer to use. The Print... will start the printing process to start printing the help file. This will normally bring up the printers dialog box. The Print Preview does just that, it can also be used for printing. The Exit does the same thing as pressing the escape key; the Help form is removed.

This form also has a “F3 Find:” button, a “F2 Up” button and a small text box for entering a string of characters to find. Pressing F3 is the same as clicking the “F3 Find:” button, the text in the small text box is searched for. Pressing F2 is the same as clicking the “F2 Up” button, same as F3 except the search is done from the current location towards the top of the large text box.

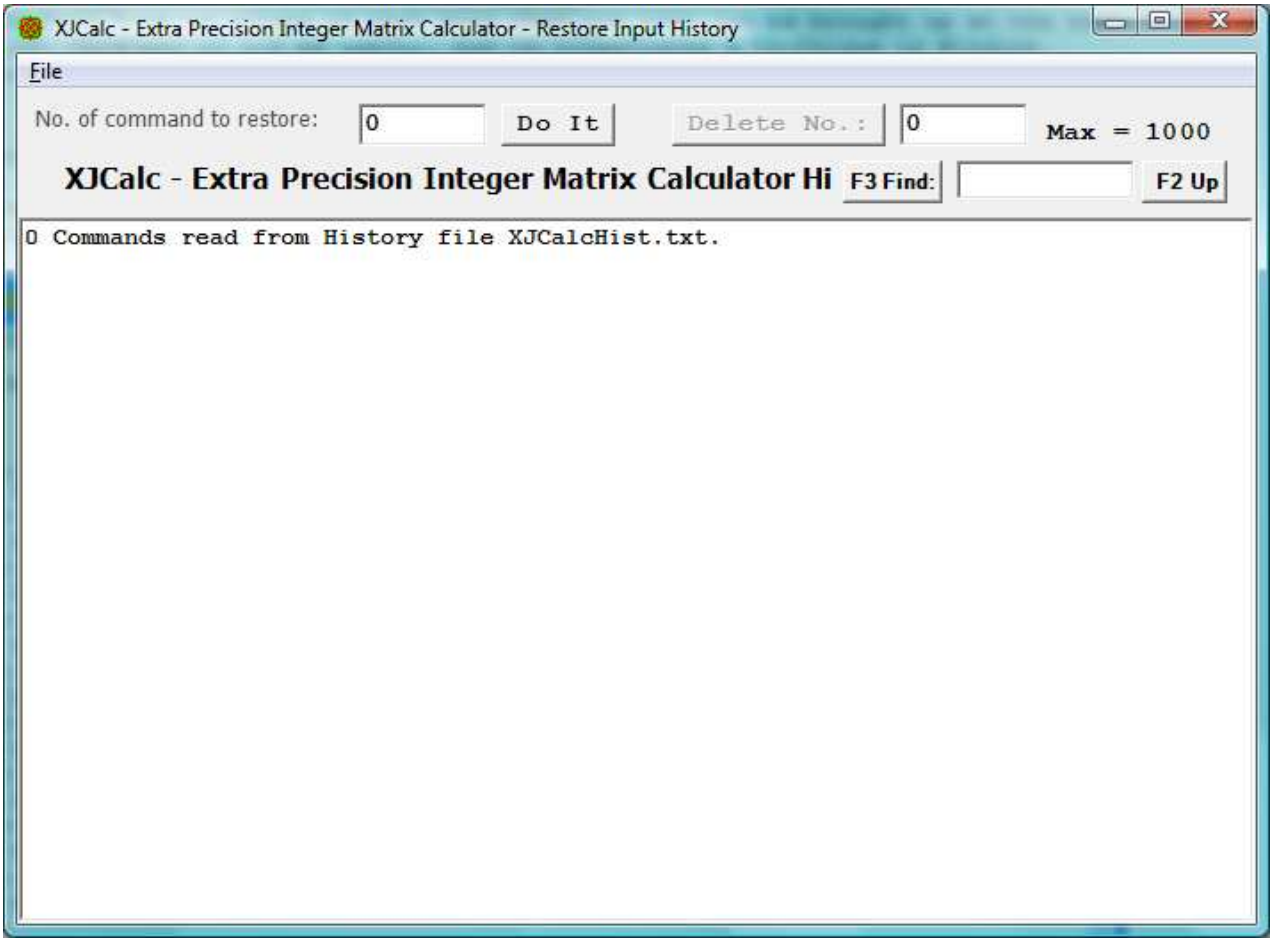
If during a find the text is not found, the sound from NotFound.wav is heard. If it is found, but only by starting over from the other end of the text, the sound from Wrap.wav is heard. This sound process uses the system file winmm.dll. If that file cannot be found, there will be no sound.

A short beep is also heard when the text is found. The beep is 277Hz = middle C# for a normal search and 554Hz one octave above for a search up. The beep process uses the system file kernel32.dll. If that file cannot be found, there will be no beep.

The Help form is unique in that more than one copy can be brought up at the same time. This may or not be useful, but it illustrates a technique in Windows programming.

Restore Input History form -

The Restore Input History form is displayed when function key F7 is pressed or the "Restore Input" command button is selected on the Run form:



The history of up to "Max Commands in History" previous operator entries are saved and can be retrieved by this form

While this form is up and the cursor is in the output text box, use the Up, Down, PgUp, and PgDn keys and the keypad + and - keys to select an entry and then use the Enter key or space bar to accept it. The command accepted will be put into the Command entry text box on the Run form, and there it can be edited before it is executed. The ESC key will remove the History form without changing the Command input text box.

The Ins key will toggle the locked status of the entry containing the cursor and move the cursor to the next higher command. When an entry is locked it cannot be deleted or scrolled off the bottom of the list. A # will be displayed to the left of a locked entry.

The Del key will delete the entry containing the cursor, if it is not locked, and move to the next higher command. The command numbers in the two command number entry text boxes are updated as commands are locked, unlocked, deleted or scrolled to by the + and - keypad keys. The - key is up and + is down due to layout of the keypad, - above +. Only the keypad + and - keys are used here.

The mouse can also be used. A left mouse click will select a command and a right mouse click will accept it.

The "Number of command to restore:" text box and the "Do It" command button can be used to select a command to be restored to the command input text box. The "Delete Command No." command button and its text box can be used to delete a given command by number. These controls act in the same way. Its text box can be edited and a click on the command button will accept the entry. A space at the right of the text or a Return/Enter key will also accept it.

When the Delete Command No.: button is selected the command referenced is deleted if it is not locked. If locked the command number field will be increased to point to the next command or a set to 1 if at max number.

The Restore input History form has a file menu with Clear History, Restore history, Save History, and Exit.

Selecting Clear History will remove all commands saved in memory. This is the same as the ClearHist command.

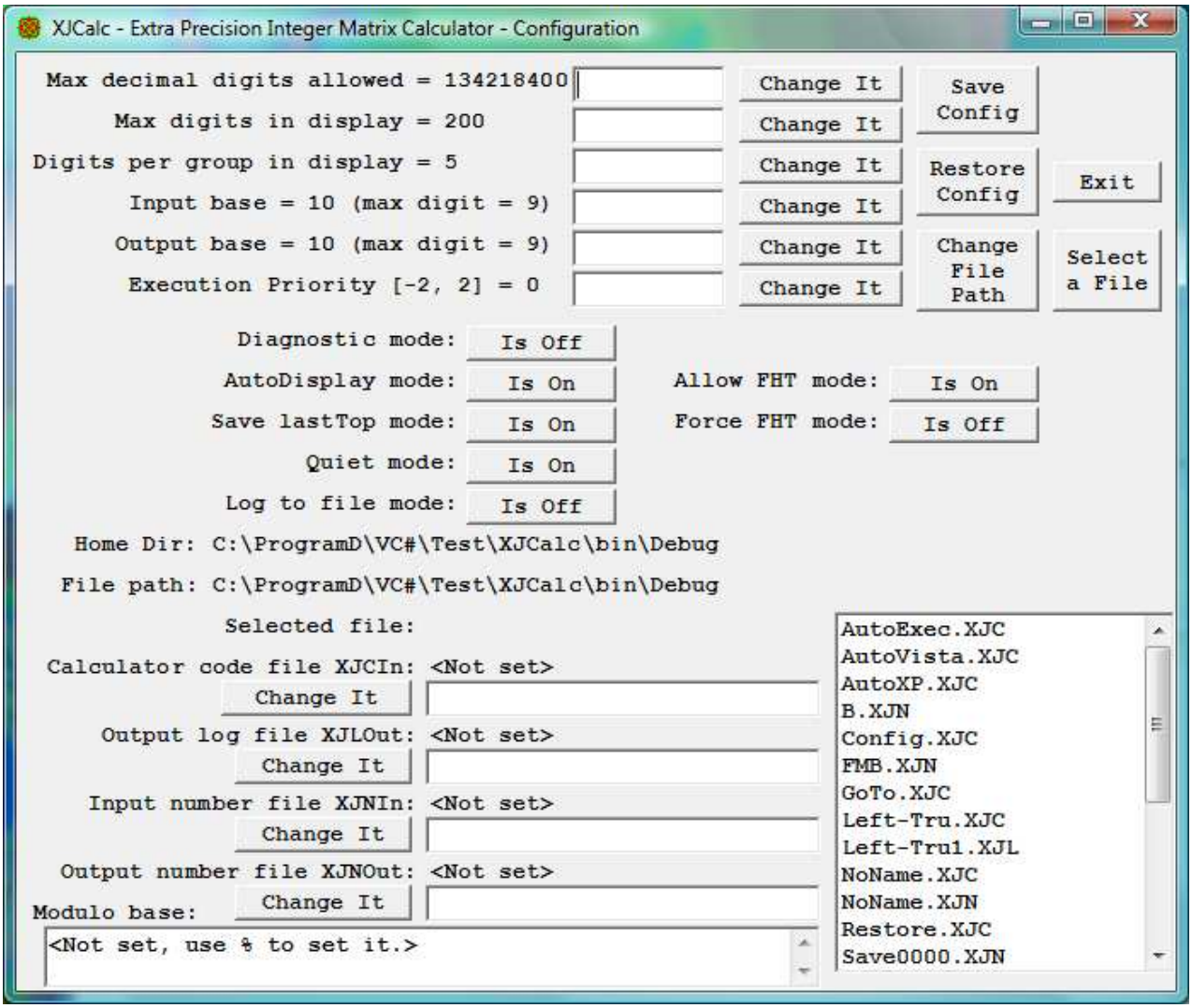
Selecting Restore History will read in the file XJCalcHist.txt and add its commands to the ones in memory. This is the same as the [command. Duplicate commands are deleted as the new copy is entered, but the lock state will be as set on the copy of the command in memory.

Selecting Save History will write all of the current command history to file XJCalcHist.txt. This is the same as the] command. If a command history file already exists, it will be renamed XJCalcHist.Bak.

Selecting Exit does the expected, same as the escape key.

Configuration form -

The Configuration form is displayed when function key F6 is pressed or the "Configuration" command button is selected on the Run form:



This form allows you to change:

Max Decimal digits allowed, same as the M command or SetMax(X) command.
Max digits in display, same as the SetD(X) command.
Digits per group in display, same as the G command.
Input base = xx (max digit = x), same as the BaseI(X) command
Output base = xx (max digit = x), same as the BaseO(X) command
Execution Priority [-2, 2] = {p}, same as the Pri(X) command
Diagnostic mode, a combination of Diag(X) and Time commands.
AutoDisplay mode, same as the A command.
Save LastTop mode, same as the SaveTop(X) command.
Quiet mode, same as Quiet(X) command.
Log to file mode, same as the LogScreen(X) command.
Allow FHT mode, same as the AllowFHT(X) command.
Force FHT mode, same as the ForceFHT(X) command.
Name of Calculator code file XJCIn, same as the XJCIn(F) command.
Name of Output Log file XJLOut, same as the XJLOut(F) command.
Name of input number file XJNIn, same as the XJNIn(F) command.
Name of output number file XJNOut, same as the XJNOut(F) command.

It also shows the Home Directory, File path, Modulo Base, and the name of the log file if the LogScreen mode is on. The Browse For Folder form can be brought up by selecting the "Change File Path" command button, and the Select a file form can be brought up by selecting "Select a File" command button. The Save Config button is the same as the > command to save the configuration to file Config.XJC. The Restore Config button is the same as the < command to restore the configuration.

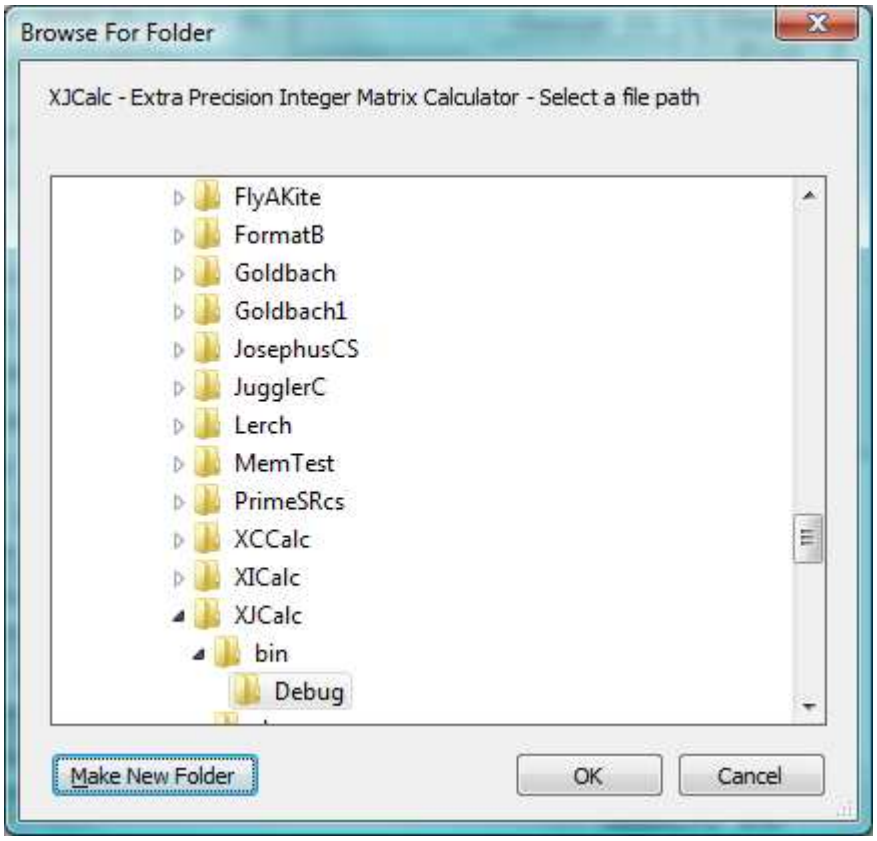
All ten “Change It” buttons act in the same way. Clicking it when its text box is empty will bring up the current value. It can be edited, and a second click with the text box not empty will accept the entry. A space at the right or a Return/Enter key will also accept it.

The seven modes can be changed/toggled by clinking the corresponding button. If a mode is on, the button will says “Is On”, then clicking it will turn the mode off and the button will change to “Is Off”. If a mode is off, the button will says “Is Off”, then clicking it will turn the mode on and the button will change to “Is On”. The Diagnostic mode differs, it has four states, Is Off, Time, Diag, and Debug.

The Log to file mode and Name of output log file XJLOut interact in that if the file name is changed while a log file is open, the file name of the open file will not change until the log file is closed and reopened.

Browse For Folder form -

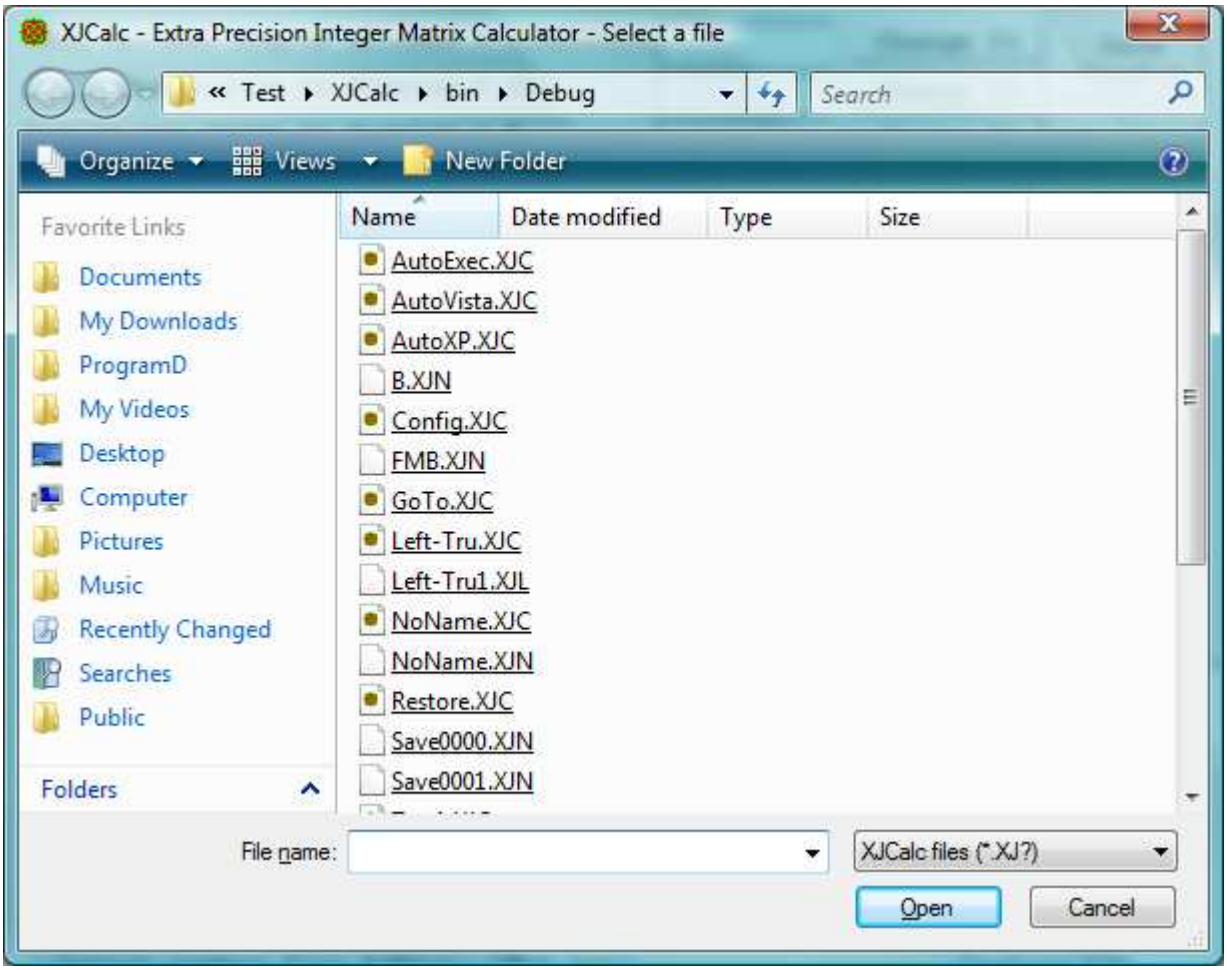
The Browse For Folder form is displayed when the “Change File Path” command button is selected on the Configuration form:



To change the current directory to be used by the H, I, J, W, ReadN(F), and Run(F) commands, use this form to select a path. This form is resizable as is the Run, Help, History, and Select a file forms.

Select a File form -

The Select a file form is displayed when the “Select a File” command button is selected on the Configuration form:



This provides a way to browse and select a file. If a file is select, it can then be used to set a file name on the Configuration form by clicking one of the file name "Change It" buttons. The first click enters the file name in the text box and a second click will accept it. This form can also be used to change the file path.

The list box in the lower right hand corner of the Configuration form is a list of the files in the currently selected file path. The wild card selection for these files (*.XJ?, *.txt, or *.*) is the same as last used on the Select a file form.

The distribution files can be downloaded from my website:

http://www.geocities.com/hjsmithh/

in the **Files to Download** section

http://www.geocities.com/hjsmithh/download.html#XJCalc.

When you install the program using the distribution file XJCalc32?.exe or XJCalc32?.zip, a folder is created with 4 subfolders:

```
├──XJCalc 3.2
│   ├──doc
│   ├──run
│   ├──src
│   └──test
```

The main folder has two files sseexec.dat and SSEun.dat. These are needed to be able to uninstall the program

The doc subfolder has the following 4 files:

```
00.txt     (A short list of features)
Fixes.txt  (A list of features and fixes added to each version)
XJCalc.doc (Microsoft Word file)
XJCalc.txt (Plain ASCII text copy of the .doc file without graphics)
```

There is also available on my website a copy of the .doc file in an Adobe Acrobat Reader file XJCalc.pdf.

The run subfolder has the following 9 files

```
00.txt
AutoExec.XJC
AutoVista.XJC
AutoXP.XJC
NotFound.wav
Wrap.wav
XJCalc.exe
XJCalc.exe.config
XJCalcHelp.txt
```

The .exe file is executed from there.

The src subfolder has all the source files needed for development:

```
00Note.txt
AboutForm.cs
AboutForm.resx
app.config
AssemblyInfo.cs
Calc.cs
Common.cs
ConfigForm.cs
ConfigForm.resx
COPYING.txt
cs.ico
FHTMult.cs
Global.cs
harry.jpg
HelpForm.cs
HelpForm.resx
HistoryForm.cs
HistoryForm.resx
MultiID.cs
MultiJD.cs
NPrimes.cs
PrimeFA.cs
RunForm.cs
RunForm.resx
StartupForm.cs
StartupForm.resx
XICalLst.cs
XIMult.cs
XJCalc.csproj
XJCalc.sln
XJCalc.suo
XJCalc.csproj.user
XJMult.cs
```

The test subfolder has the files I use for testing the program. They are:

```
00.txt
AutoExec.XJC
AutoVista.XJC
AutoXP.XJC
B.XJN
Config.XJC
FMB.XJN
GoTo.XJC
Left-Tru.XJC
Left-Tru.XJL
NoName.XJC
NoName.XJN
NotFound.wav
Restore.XJC
Save0000.XJN
Save0001.XJN
Test.Bat
Test1.XJC
Test1.XJL
Test2.XJC
Test2.XJL
Test3.XJC
Test3.XJL
Test4.XJC
Test4.XJL
Test5.XJC
Test5.XJL
WhatForTst.txt
Wrap.wav
XJCalc.exe.config
XJCalcHelp.txt
XJCalcHist.Bak
XJCalcHist.txt
XJCalcPTab.Bak
XJCalcPTab.txt
XJInfo.txt
```

The program can generate the following files:

```
Config.XJC
FMB.XJN
NoName.XJC
NoName.XJL
NoName.XJN
Restore.XJC
Savexxxx.XJN
XJCalcHist.Bak
XJCalcHist.txt
XJCalcPTab.Bak
XJCalcPTab.txt
```

The end -

Report any errors by sending me a letter, an e-mail or call me at my home phone.

-Harry

Harry J. Smith
19628 Via Monte Dr.
Saratoga, CA 95070-4522, USA

Home Phone: 1 408 741-0406
E-mail: hjsmithh@sbcglobal.net
Website: http://www.geocities.com/hjsmithh/