Arbitrary Precision Calculator (APCalc)

based on “Calc - C-style arbitrary precision calculator”
(2.12.2.2 with fixes up to 2.12.4.3)
by Landon Curt Noll et. al.
http://isthe.com/chongo/tech/comp/calc/

The original Calc was modified by eliminating all of the file-based and terminal-based i/o routines. All communication with the resulting Calc library is accomplished by function calls. This library was developed (along with a primitive user interface) in Bloodshed Dev-C++ version 4.9.9.2.

APCalc is implemented as a Qt-based user interface to the Calc library. This user interface approximates a blend of the features of the HP-42s and the HP-48, but without graphics, statistical, or CAS functionality. The current interface was developed in Qt Creator 2.0.1 "based on Qt 4.7.0 (32 bit)".

APCalc menus are **File**, **Mode**, **Solve**, **Equation** (Algebraic mode only), **Unit**, and **Constant** (more below).

APCalc has three basic modes, RPN, Algebraic (Algebraic Interactive), Program, and Solve. By default, an ini file named “APCalculator.ini” in the executable’s directory is applied to initialize APCalc modes and “built-in” function definitions. In the shortcut’s properties, you may specify another ini file after the path to the executable file.

Numeric Value Entry Boxes

Value entry boxes supports the usual copy, cut, and paste operations, and supports the keyboard, except that only legal values may be entered. These are: binary ('0b'/'0B' followed by one or more of '0' and'1'), decimal ('-d', '-d.d', '-.d', '-dE-d', '-d.dE-d', '-.dE-d', '-di', '-d.di', '-.di', '-dE-di', '-d.dE-di', '-.dE-di', where d represents one or more decimal digits and the minus signs are optional), and hexadecimal ('0x'/'0X' followed by one or more of '0' through '9' and 'A' through 'F'). A leading zero always signifies a non-decimal base, either binary or hexadecimal, and must be followed by 'b'/'B' (binary) or 'x'/'X' (hexadecimal). Therefore, the first (non-fraction) decimal digit in a decimal value may not be '0', except when the integer part of the decimal value equals zero, for example, a zero decimal value or a leading zero followed by the decimal point.

Starting with an empty value entry box, if you attempt to enter the value of i by clicking the i button, '1i' will be entered. If you attempt to enter a bare exponent by clicking the E button, '1E' will be entered. More generally, the value entry boxes will attempt to prevent the entry of anything but legal values as defined above.

The value entry boxes (in all modes, RPN, Algebraic, and Solve) dynamically display decimal values with the customary thousands digit grouping (similar to the HP-42s). Hexadecimal and binary values are also grouped, but in groups of four digits. Thousands digit grouping for decimal values (and the analogous grouping of hexadecimal and binary values) is also applied to the value stack and memory register data display, as well as the display of solve variable data.

APCalc RPN Mode

In RPN mode, there are (top to bottom) a value stack display, a numeric value entry box, and a 6-row by 7-column button grid.

The value stack display shows 10 entries, but the actual stored value stack is effectively unlimited. Only the most recent 10 are displayed. Clicking on a value in the value stack display loads that entry into the value entry box.

The value stack display may also show the values of 10 memory registers, A to J. The memory registers can be loaded by **STO**, optionally followed by +, -, *, or /, and clicking on one of the registers. If no optional +, -, *, or / is used, the register will be loaded with either the contents of the value entry box (if any) or the value held by the bottom value stack entry. If the optional +, -, *, or / is used, the same value will be added to, subtracted from, multiplied with, or divided into the selected register and the result will be stored in the register.

Similarly, **RCL**, optionally followed by +, -, *, or /, and clicking on a memory register, will either push the register content onto the value stack, or perform the corresponding operation on the bottom value stack entry.

To view the memory registers, click **STO**, and click **STO** again to return to the value stack view.

RPN mode buttons:

In RPN mode, the **SHFT** button shifts to another set of functions, but only for the next use of one of the shifted buttons, after which the button set returns to the unshifted button set.

With the exception of **SHFT**, **STO**, and **RCL**, in RPN mode, all of the buttons marked in bold text will affect the stack: by operating on one or more value stack values (e.g. sin, +, +/-, **Swap**), by pushing a value on the stack (e.g. Pi and e), by popping a value off the stack (e.g.Drop), or by reverting to a previous/next state of the stack (e.g. **Undo**, **Redo**). The buttons marked in light text only affect the value entry box. Other buttons can enter text into the value entry box: Alt-Hex enters '0b' (to commence binary value entry), Hex enters '0x' (to commence hexadecimal value entry), and A through F enter 'A' through 'F' (for hexadecimal value entry). The keyboard Enter key is equivalent to the **Ent** button.

When a button that operates on the stack is clicked, and a value is present in the value entry box, that value will typically be pushed on the stack before the operation is executed.

The **Undo/Redo** buttons will restore the state of the previous stack/memory of the RPN calculator. In this manner, the RPN calculator can be completely reset to its stack/memory state prior to up to 1000 previous calculation steps. You may also use this feature to review previous calculations and their results. In this case you should be aware that the visible state of the RPN calculator properly represents it actual internal state, so that further calculation steps will “lose” the formerly succeeding steps.

Unit Menu (unit conversions):
The unit conversions found on the unit menu push the selected conversion factor onto the Value Stack. For example, to convert 17 inches to centimeters, type '17', select Unit, Length, inches to centimeters from the **Unit** menu. Now click the multiply button which leaves the converted value on the bottom of the Value Stack.

Constant Menu (the values of various mathematical and physical conatants):
This menu works similarly to the **Unit** Menu.

APCalc Algebraic (Algebraic Interactive) Mode

Algebraic and Algebraic Interactive modes are essentially equivalent, except that in Algebraic Interactive mode, the value box interactively displays the value of the expression being typed (if syntactically correct) as it is being typed. In Algebraic mode, the expression is only evaluated (and its value displayed) when the **Eval** button is clicked.

In Algebraic mode, there are (top to bottom) an expression evaluation value/result box, an expression entry box, a numeric value entry box, , and a 7-row by 7-column button set. The Escape key forces an evaluation (as does the **Eval** button). The Insert key takes you to the numeric value entry box, whose rules are detailed above. When the keyboard Enter key is pressed, the entered value will be inserted into the expression/program entry box at the current insertion point.

Algebraic mode buttons:
Two shift buttons, **Trig** and **Math**, shift to aternate button sets, but only for the next use of one of the shifted buttons, after which the button set returns to the unshifted button set. The Home button will also return to the unshifted button set.
.

APCalc Program Mode

In Program mode, there are (top to bottom) an expression/program evaluation value/result box, an expression/program entry box, and a 6-row by 6-column button set. The Escape key forces an evaluation (as does the **Eval** button).

Program mode buttons:
The button grid can be shifted through three different button sets. In Program mode, the current button set is sticky and the bottom left button identifies the topic of the button set.

APCalc History

APCalc keeps a record of all calculation operations performed. This record can be exported in text form (*.his) using **Export History…** on the **File** menu. This record can be restored using **Import History…** on the **File** menu. In RPN mode, the state of APCalc (with the exception of settings on the **Mode** menu) can be restored to the state saved in an exported history by first clearing history (**Clear History** on the **File** menu) and then importing the saved history, and finally clicking the **Undo** button.

Recorded history can save up to 1000 states. Each RPN mode operation which changes the stack or memory registers (except undo/redo) counts as one state. Each press of the **Eval** button (or two successive newlines) in Algebraic mode counts as one state. When the maximum number of saved states has been reached, each new saved state causes the earliest saved state to be lost and the new state saved.

The history file (*.his) can save any combination of RPN, Algebraic, or Program operations, in the order of execution. See history file examples below.

APCalc Solve Mode
Entering parameter data values for the Solve variables follows the same rules as entering numerical values into a numeric value entry box as detailed above.

A solve file (*.slv) can contain user defined solves, and can be imported with **Solve Menu, Import…** An example equivalent to **Solve Menu, Time Value of Money…** is shown below. All numeric values must use the period radix mark and may have no digit separators. All solvable variables must be marked with '/*?*/'. For example, it would not be possible to solve for `PeriodsPerYr` (number of payments per year) because this must be an integer value, e.g. 1, 2, 4, 6, or typically 12. `Pbegin` indicates whether payments are made at the beginning of the period (true = 1) or the end of the period (false = 0).

APCalc Menus

File

Open INI... – Apply saved modes of APCalc from a file.
Save INI As... – Save current modes of APCalc to a file.

Import Defines...
Export Defines...
Clear Defines

Import History...
Export History...
Clear History

Exit

Mode

Algebraic – Forces APCalc into Algebraic mode.
Algebraic Interactive – Forces APCalc into Algebraic Interactive mode.
Program – Forces APCalc into Program mode.
RPN – Forces APCalc into RPN mode.

Period Radix Mark – Values displayed with period radix mark / comma separator.
Comma Radix Mark – Values displayed with comma radix mark / period separator.

Degrees – Angle values displayed and interpreted as degrees.
Radians – Angle values displayed and interpreted as radians.

Decimal
Scientific
Hexadecimal – Displays integer values as hexadecimal, with digits grouped by fours.
Binary – Displays integer values as binary, with digits grouped by fours.
Decimal/Hexadecimal – Displays integer values as hexadecimal in a comment.
Decimal/Binary – Displays integer values as binary in a comment.

Recall Eval
Recall Eval and Value

Solve (forces APCalc into Solve Mode)

- **Import** – Import solve file (*.slv), example below equivalent to TVM menu item.
- ---
- **Finance**
  - **Time Value of Money** – TVM calculations.
  - **TVM (Odd Period)** – TVM w/ days before first payment.
  - ---
  - **Advance Payments**
  - ---
  - **Discounted Notes Price**
  - **Discounted Notes Yield**
  - ---
  - **Interest Rate Conversion**
  - **Int Rate Conv (Continuous)**
- **Dates**
  - **Elapsed Days**
  - **Add Days**
- **Geometry**
  - **Side of Right Triangle**
  - **Area of Circle**
  - **Volume of Sphere**

Equation (Algebraic modes only)

- **Math**
  - **Normal Probability**
- **Geometry**
  - **Circumference of Circle**
  - **Area of Circle**
  - **Volume of Sphere**

Unit (conversion factors)

- **Length**
  - **centimeters to inches**
  - **meters to feet**
  - **meters to yards**
  - **kilometers to miles**
  - ---
  - **inches to centimeters**
  - **feet to meters**
  - **yards to meters**
  - **miles to kilometers**
  - ---
  - **inches to feet**
  - **inches to yards**
  - **feet to miles**
  - **yards to miles**
  - **feet to inches**
  - **yards to inches**
  - **miles to feet**
  - **miles to yards**
- **Area**
  - **sq millimeters to sq inches**
  - **sq centimeters to sq inches**
  - **sq centimeters to sq feet**
  - **sq meters to sq feet**
  - **sq meters to sq yards**
  - **sq meters to acres**
  - ---
  - **sq inches to sq millimeters**
  - **sq inches to sq centimeters**
  - **sq feet to sq centimeters**
  - **sq feet to sq meters**
  - **sq yards to sq meters**
  - **acres to sq meters**
  - ---
  - **sq inches to sq feet**
  - **sq inches to sq yards**
  - **sq feet to acres**
  - **sq yards to acres**
  - **acres to sq yards**
  - **acres to sq feet**
  - **sq yards to sq inches**
  - **sq feet to sq inches**

Unit (conversion factors)

- **Volume**
  - **cu centimeters to cu inches**
  - **liters to cu feet**
  - **cu meters to cu inches**
  - **cu meters to cu feet**
  - **cu meters to cu yards**
  - ---
  - **cu inches to cu centimeters**
  - **cu inches to cu meters**
  - **cu feet to liters**
  - **cu feet to cu meters**
  - **cu yards to cu meters**
  - ---
  - **cu inches to cu feet**
  - **cu inches to cu yards**
  - **cu feet to cu inches**
  - **cu yards to cu inches**
- **Capacity**
  - **liters to USquart**
  - **liters to USgallon**
  - **cu meters to USgallon**
  - ---
  - **USquart to liters**
  - **USgallon to liters**
  - **USgallon to cu meters**
- **Mass/Force**
  - **kg to pound (av)**
  - ---
  - **pound (av) to kg**
- **Energy/Power**
  - **watts to horsepower**
  - ---
  - **horsepower to watts**
- **Angle**
  - **arcsec to degrees**
  - **arcmin to degrees**
  - **grads to degrees**
  - ---
  - **degrees to arcsec**
  - **degrees to arcmin**
  - **degrees to grads**

Unit (conversion factors)

- **Time/Speed**
  - **seconds to years**
  - **years to seconds**
  - ---
  - **miles/hour to knots**
  - **miles/hour to meters/sec**
  - **meters/sec to miles/hour**
  - **knots to miles/hour**
- **Temperature**
  - **degrees C to degrees F**
  - ---
  - **degrees F to degrees C**

Constant (values)

- **Physics**
  - **Speed of Light**
  - **Gravitational Constant**
  - **Acceleration of Gravity (m/s)**
  - **Acceleration of Gravity (ft/s)**
  - ---
  - **Charge of Electron**
  - **Mass of Electron**
  - **Mass of Proton**
  - **Mass of Neutron**
  - ---
  - **Avogadro Constant**
  - **Planck Constant**
- **Geography**
  - **Radius of Earth (equat, km)**
  - **Radius of Earth (polar, km)**
  - ---
  - **Radius of Earth (equat, mi)**
  - **Radius of Earth (polar, mi)**

Constant (values)

- **Orders of Magnitude**
  - **yotta**
  - **zetta**
  - **exa**
  - **peta**
  - **tera**
  - **giga**
  - **mega**
  - **kilo**
  - **hecto**
  - ---
  - **centi**
  - **milli**
  - **micro**
  - **nano**
  - **pico**
  - **femto**
  - **atto**
  - **zepto**
  - **yocto**

- **Data Magnitude**
  - **kibi**
  - **mebi**
  - **gibi**
  - **tebi**
  - **pebi**
  - **exbi**
  - **zebi**
  - **yobi**

APCalc functions

Button	**Description** (where x is stack value 0 and y is stack value 1)
abs	Absolute value. Returns |x|.
acos	Arc cosine. Returns cos^-1 x.
acot	Arc cotangent
acsc	Arc cosecant
acsh	Arc hyperbolic cosine. Returns cosh^-1 x.
asec	Arc secant
asin	Arc sine. Returns sin^-1 x.
asnh	Arc hyperbolic sine. Returns sinh^-1 x.
atan	Arc tangent. Returns tan^-1 x.
atan2	Returns the angle t such that -pi < t <= pi and x = r * cos(t), y = r * sin(t).
atnh	Arc hyperbolic tangent. Returns tanh^-1 x.
bAnd	Bitwise AND. Returns x Bitwise AND y.
bNot	Bitwise NOT. Returns 32 bit Bitwise NOT(x) equivalent to x XOR 0xFFFFFFFF.
bOr	Bitwise OR. Returns x Bitwise OR y.
bXor	Bitwise XOR (exclusive OR). Returns x Bitwise XOR y.
cAbs	The Absolute value (radus) of an imaginary number. Returns sqrt(re(x)^2 + im(x)^2).
cArg	Argument (the angle or phase) of a complex number.
cPolr	Returns x = cArg (angle) and y = cAbs (radius) from a complex number.
ceil	Smallest integer greater than or equal to number
comb	Combinations of y items taken x at a time = y! / [x!(y-x)!]
cos	Cosine. Returns cos(x) where x may be complex.
cosh	Hyperbolic cosine. Returns cosh(x).
cot	Cotangent
csc	Cosecant
c»r	Complex to Real. Returns x = re(x), y = im(x).
den	Denominator of fraction
digit	Digit at specified decimal place of number
Ent	Push value in value entry box onto stack.
e	Base of the natural logarithm.
e^x	Natural exponential. Returns e^x.
fact	Factorial. Returns x!.
fib	Fibonacci number F(n)
floor	Greatest integer less than or equal to number
fp	Returns the fractional part of x.
gcd	Greatest common divisor, gcd(x1, x2, x3,…x1024)
im	Returns imaginary part of x as real number.
ip	Returns the integer part of x
intDiv	Integer division
lcm	Least common multiple, lcm(x1, x2, x3,…x1024)
ln	Natural logarithm. Returns ln(x).
log	Common logarithm. Returns log10(x).
log2	Returns log base 2 of x.

logx	Returns log base x of y.
max	Maximum of, max(x1, x2, x3,…x1024)
min	Minimum of, min(x1, x2, x3,…x1024)
nrnd	Normally (Gaussian) distributed random numbers, using Box-Muller.
num	Numerator of fraction
perm	Permutations of y items taken x at a time. Returns y!/(y - x)!
pi	Put an approximation of pi into the x-register (3.14159265359.....).
Polr	Returns a complex number from x = cArg (angle) and y = cAbs (radius).
rand	Returns a random number (0 = x < 1)
RCL	Recall data from memory register into x.
RCL+	Recall data from memory register and add it to the contents of x.
RCL-	Recall data from memory register and subtract it from the contents of x.
RCL×	Recall data from memory register and multiply it by the contents of x.
RCL÷	Recall data from memory register and divide it into the contents of x.
Redo	Recall next stack and memory register contents.
re	Returns real part of x.
r»c	Real to Complex. Returns x + (y * i).
round	Round value a to b number of decimal places
sec	Secant
seed	Store a seed for the random number generator.
sin	Sine. Returns sin(x) where x may be complex.
sinh	Hyperbolic sine. Returns sinh(x).
ST0	Store a copy of x into a memory register.
STO+	Adds x to a memory register.
STO-	Subtracts x from a memory register.
ST0×	Multiplies a memory register by x.
ST0÷	Divides a memory register by x.
tan	Returns tan(x).
tanh	Returns tanh(x).
trunc	Truncate a to b number of decimal places
Swap	Swaps the contents of the x- and y-registers.
Undo	Recall previous stack and memory register contents.
x²	Returns x^2.
√	Returns square root of x.
y^x	Returns y^x.
x√y	Returns y^(1/x).
y // x	Returns the integer quotient for y / x.
y % x	Returns the remainder for y / x.
1/x	Returns 1/x.
10^x	Returns 10^x.
+ (on right)	Returns y + x.
- (on right)	Returns y - x.
× (on right)	Returns x × y.
÷ (on right)	Returns y / x.
+/-	Change the sign of the number in x.

»deg	To degrees. Convert an angle-value from radians to degrees. Returns x×(180/p).
»rad	To radians. Converts a angle value in degrees to radians. Returns x×(p/180).
%	Percent. Returns (x × y) / 100. (Leaves the y value in the y-register.)
%ch	Percent change. Returns (x - y) × (100 / y).
%t	Percent of total. Returns 100 × y / x.

(**In light-face type)**

0-9, ., -	Entering numerical values, potentially negative.
Hex, A-F	Initiate hexadecimal numeric value and hexadecimal digits 10-15.
At-Hex	Initiate binary numeric value.
Ex, I, -	Entering numerical exponents, potentially negative, and imaginary values.
Del	Deletes the character prior to the insertion point (text cursor).

Notes on APCalc functions

Only sin(x) and cos(x) accept complex values. All other trigonometric (tan(x), cot(x), asin(x), acos(x), etc.) and hyperbolic (sinh(x), cosh(x), asinh(x), etc.) functions require real values.

Algebraic Mode Defines File (*.def) Example

```
define spherevolume(radius) = ((4*pi/3) * (radius^3));
define circlearea(radius) = (pi * (radius^2));
define step(x) { if(x < 2) return x^1;
                 else if (x < 3) return x^2;
                 else return x^3; };
```

After importing this defines file, evaluating `spherevolume(5)` will display:

```
523.59877559829887307710723054658381403286 67
```

evaluating `circlearea(5)` will display:

```
78.53981633974483096156608458198757210493
```

and evaluating `step(1);step(2);step(3)` will display:

```
1
4
27
```

RPN Mode History File (*.his) Format
(all values have period radix mark and no digit separator):

```
/* Begin RPN 2010-12-23 19:56:44 */
/* Eval */
/* Value */
free(rpnstack);rpnstack=list();
push(rpnstack, 123456789);

free(rpnmemory);rpnmemory = list(0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
{delete(rpnmemory, 9); insert(rpnmemory, 9, 0);};
{delete(rpnmemory, 8); insert(rpnmemory, 8, 0);};
{delete(rpnmemory, 7); insert(rpnmemory, 7, 0);};
{delete(rpnmemory, 6); insert(rpnmemory, 6, 0);};
{delete(rpnmemory, 5); insert(rpnmemory, 5, 0);};
{delete(rpnmemory, 4); insert(rpnmemory, 4, 0);};
{delete(rpnmemory, 3); insert(rpnmemory, 3, 0);};
{delete(rpnmemory, 2); insert(rpnmemory, 2, 0);};
{delete(rpnmemory, 1); insert(rpnmemory, 1, 0);};
{delete(rpnmemory, 0); insert(rpnmemory, 0, 0);};
/* End */

/* Begin RPN 2010-12-23 19:56:47 */
/* Eval */
/* Value */
free(rpnstack);rpnstack=list();
push(rpnstack, 123456789);
push(rpnstack, 456789);

free(rpnmemory);rpnmemory = list(0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
{delete(rpnmemory, 9); insert(rpnmemory, 9, 0);};
{delete(rpnmemory, 8); insert(rpnmemory, 8, 0);};
{delete(rpnmemory, 7); insert(rpnmemory, 7, 0);};
{delete(rpnmemory, 6); insert(rpnmemory, 6, 0);};
{delete(rpnmemory, 5); insert(rpnmemory, 5, 0);};
{delete(rpnmemory, 4); insert(rpnmemory, 4, 0);};
{delete(rpnmemory, 3); insert(rpnmemory, 3, 0);};
{delete(rpnmemory, 2); insert(rpnmemory, 2, 0);};
{delete(rpnmemory, 1); insert(rpnmemory, 1, 0);};
{delete(rpnmemory, 0); insert(rpnmemory, 0, 0);};
/* End */
```

```
/* Begin RPN 2010-12-23 19:56:49 */
/* Eval */
/* Value */
free(rpnstack);rpnstack=list();
push(rpnstack, 123456789);
push(rpnstack, 456789);
push(rpnstack, 789);

free(rpnmemory);rpnmemory = list(0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
{delete(rpnmemory, 9); insert(rpnmemory, 9, 0);};
{delete(rpnmemory, 8); insert(rpnmemory, 8, 0);};
{delete(rpnmemory, 7); insert(rpnmemory, 7, 0);};
{delete(rpnmemory, 6); insert(rpnmemory, 6, 0);};
{delete(rpnmemory, 5); insert(rpnmemory, 5, 0);};
{delete(rpnmemory, 4); insert(rpnmemory, 4, 0);};
{delete(rpnmemory, 3); insert(rpnmemory, 3, 0);};
{delete(rpnmemory, 2); insert(rpnmemory, 2, 0);};
{delete(rpnmemory, 1); insert(rpnmemory, 1, 0);};
{delete(rpnmemory, 0); insert(rpnmemory, 0, 0);};
/* End */

/* Begin RPN 2010-12-23 19:56:52 */
/* Eval */
/* Value */
free(rpnstack);rpnstack=list();
push(rpnstack, 123456789);
push(rpnstack, 456789);
push(rpnstack, 789);

free(rpnmemory);rpnmemory = list(0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
{delete(rpnmemory, 9); insert(rpnmemory, 9, 0);};
{delete(rpnmemory, 8); insert(rpnmemory, 8, 0);};
{delete(rpnmemory, 7); insert(rpnmemory, 7, 0);};
{delete(rpnmemory, 6); insert(rpnmemory, 6, 0);};
{delete(rpnmemory, 5); insert(rpnmemory, 5, 0);};
{delete(rpnmemory, 4); insert(rpnmemory, 4, 0);};
{delete(rpnmemory, 3); insert(rpnmemory, 3, 0);};
{delete(rpnmemory, 2); insert(rpnmemory, 2, 0);};
{delete(rpnmemory, 1); insert(rpnmemory, 1, 0);};
{delete(rpnmemory, 0); insert(rpnmemory, 0, 789);};
/* End */
```

```
/* Begin RPN 2010-12-23 19:56:57 */
/* Eval */
/* Value */
free(rpnstack);rpnstack=list();
push(rpnstack, 123456789);
push(rpnstack, 456789);
push(rpnstack, 789);
push(rpnstack, 456);

free(rpnmemory);rpnmemory = list(0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
{delete(rpnmemory, 9); insert(rpnmemory, 9, 0);};
{delete(rpnmemory, 8); insert(rpnmemory, 8, 0);};
{delete(rpnmemory, 7); insert(rpnmemory, 7, 0);};
{delete(rpnmemory, 6); insert(rpnmemory, 6, 0);};
{delete(rpnmemory, 5); insert(rpnmemory, 5, 0);};
{delete(rpnmemory, 4); insert(rpnmemory, 4, 0);};
{delete(rpnmemory, 3); insert(rpnmemory, 3, 0);};
{delete(rpnmemory, 2); insert(rpnmemory, 2, 0);};
{delete(rpnmemory, 1); insert(rpnmemory, 1, 0);};
{delete(rpnmemory, 0); insert(rpnmemory, 0, 789);};
/* End */

/* Begin RPN 2010-12-23 19:57:00 */
/* Eval */
/* Value */
free(rpnstack);rpnstack=list();
push(rpnstack, 123456789);
push(rpnstack, 456789);
push(rpnstack, 789);
push(rpnstack, 456);

free(rpnmemory);rpnmemory = list(0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
{delete(rpnmemory, 9); insert(rpnmemory, 9, 0);};
{delete(rpnmemory, 8); insert(rpnmemory, 8, 0);};
{delete(rpnmemory, 7); insert(rpnmemory, 7, 0);};
{delete(rpnmemory, 6); insert(rpnmemory, 6, 0);};
{delete(rpnmemory, 5); insert(rpnmemory, 5, 0);};
{delete(rpnmemory, 4); insert(rpnmemory, 4, 0);};
{delete(rpnmemory, 3); insert(rpnmemory, 3, 0);};
{delete(rpnmemory, 2); insert(rpnmemory, 2, 0);};
{delete(rpnmemory, 1); insert(rpnmemory, 1, 456);};
{delete(rpnmemory, 0); insert(rpnmemory, 0, 789);};
/* End */
```

```
/* Begin RPN 2010-12-23 19:57:04 */
/* Eval */
/* Value */
free(rpnstack);rpnstack=list();
push(rpnstack, 123456789);
push(rpnstack, 456789);
push(rpnstack, 789);
push(rpnstack, 456);
push(rpnstack, 123);

free(rpnmemory);rpnmemory = list(0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
{delete(rpnmemory, 9); insert(rpnmemory, 9, 0);};
{delete(rpnmemory, 8); insert(rpnmemory, 8, 0);};
{delete(rpnmemory, 7); insert(rpnmemory, 7, 0);};
{delete(rpnmemory, 6); insert(rpnmemory, 6, 0);};
{delete(rpnmemory, 5); insert(rpnmemory, 5, 0);};
{delete(rpnmemory, 4); insert(rpnmemory, 4, 0);};
{delete(rpnmemory, 3); insert(rpnmemory, 3, 0);};
{delete(rpnmemory, 2); insert(rpnmemory, 2, 0);};
{delete(rpnmemory, 1); insert(rpnmemory, 1, 456);};
{delete(rpnmemory, 0); insert(rpnmemory, 0, 789);};
/* End */

/* Begin RPN 2010-12-23 19:57:08 */
/* Eval */
/* Value */
free(rpnstack);rpnstack=list();
push(rpnstack, 123456789);
push(rpnstack, 456789);
push(rpnstack, 789);
push(rpnstack, 456);
push(rpnstack, 123);

free(rpnmemory);rpnmemory = list(0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
{delete(rpnmemory, 9); insert(rpnmemory, 9, 0);};
{delete(rpnmemory, 8); insert(rpnmemory, 8, 0);};
{delete(rpnmemory, 7); insert(rpnmemory, 7, 0);};
{delete(rpnmemory, 6); insert(rpnmemory, 6, 0);};
{delete(rpnmemory, 5); insert(rpnmemory, 5, 0);};
{delete(rpnmemory, 4); insert(rpnmemory, 4, 0);};
{delete(rpnmemory, 3); insert(rpnmemory, 3, 0);};
{delete(rpnmemory, 2); insert(rpnmemory, 2, 123);};
{delete(rpnmemory, 1); insert(rpnmemory, 1, 456);};
{delete(rpnmemory, 0); insert(rpnmemory, 0, 789);};
/* End */
```

Algebraic Mode History File (*.his) Format
(all values have period radix mark and no digit separator)

```
/* Begin Algebraic 2010-12-23 20:43:38 */
/* Eval */
mat abc[2,2] = {11, 12,
                          21, 22};
/* Value */

/* End */

/* Begin Algebraic 2010-12-23 20:43:53 */
/* Eval */
print(abc);
/* Value */
mat [2,2] (4 elements, 4 nonzero):
   [0,0] = 11
   [0,1] = 12
   [1,0] = 21
   [1,1] = 22
/* End */

/* Begin Algebraic 2010-12-23 20:44:06 */
/* Eval */
mat def[2,2] = {33, 34,
                          43, 44};
/* Value */

/* End */

/* Begin Algebraic 2010-12-23 20:44:18 */
/* Eval */
print(def);
/* Value */
mat [2,2] (4 elements, 4 nonzero):
   [0,0] = 33
   [0,1] = 34
   [1,0] = 43
   [1,1] = 44
/* End */
```

```
/* Begin Algebraic 2010-12-23 20:44:32 */
/* Eval */
print(abc*def);
/* Value */
mat [2,2] (4 elements, 4 nonzero):
   [0,0] = 879
   [0,1] = 902
   [1,0] = 1639
   [1,1] = 1682
/* End */
```

Solve File (*.slv) Example
(values must have period radix mark and no digit separators)
(/*?*/ marks solvable variables)

```
NominalInt = 0.0 /*?*/ ;
NumPeriods = 000 /*?*/ ;
PresentValue = 0.00 /*?*/ ;
FutureValue = 0.00 /*?*/ ;
Payment = 0.00 /*?*/ ;
PeriodsPerYr = 12;
PBegin = 0;
define tvm()
{local i; i=NominalInt/100/PeriodsPerYr;
 return PresentValue+(1+(PBegin*i))*Payment*(((1-(1+i)^(-
         NumPeriods)))/i)+(FutureValue*((1 + i)^(-
         NumPeriods)));}
solution_digits = 15;
```

Initialization File (*.ini) Example

```
<?xml version="1.0"?>
<APCalculateInitialization>
<Modes>
<Mode MenuItem="Algebraic"/>
<Mode MenuItem="Period Radix Mark"/>
<Mode MenuItem="Degrees"/>
<Mode MenuItem="Decimal"/>
<Mode MenuItem="Recall Eval"/>
</Modes>
<Define>
define abc(x,y) = x * y;
define def(ghi,jkl) { return ghi^jkl; };
</Define>
</APCalculateInitialization>
```