Rand-Emonium Software

Presents

Rand Hack Pack Vol. 1

Included in this bundle are the following software packages:

- CurtaSim which is a simulation of the Curta mechanical calculator from HackFest 2012.

- abCalcNDA which is a GS/OS RPN calculator new desk accessory from HackFest 2013.

- apple2048 which is a number sliding and matching game from HackFest 2014.

- a2sudoku which is a number puzzle game from HackFest 2015.

- a2bejeweled which is a gem matching game from HackFest 2016.

- colourGo which is a running, jumping and colour matching game from HackFest 2017.

CurtaSim

This is a simulation of the Curta mechanical calculator written for the Apple //. It should work on any Apple // that supports ProDOS 2.4.2.

A Curta looks somewhat like a pepper grinder. To replace the crank, this simulation uses a joystick.

The primary interaction is performed by "cranking" the joystick through 360 degrees. To perform a crank, start with the joystick centred. Pull the joystick towards you and then crank around a circle clockwise until it is again pulled towards you. Then release the joystick back to the centre position.

The operand is the set of digits at the top of the screen. The sliders control the operand. The result is at the bottom of the screen. The counter is a multiplicand. Below the result is a carat that points to a digit. This is the same as the carriage position on a real Curta.

The joystick operations are:

* Left/Right - Select a digit in the operand.
* Up/Down - Change the selected digit in the operand.
* Left/Right with Button 0 Down - Change the carriage position.
* Crank - Add the operand times the carriage position to the result. If the carriage position is

pointing to the hundreds position, then 100 times the operand is added to the result.

* Crank with Button 0 Down - Subtract the operand times the carriage position from the result.
* Crank with Button 1 Down - Clear the result and counter values.

These keyboard commands can be used:

* Q - Quit the simulation.
* H - Print help information.

Imagine you want to multiply 123 by 990. Perform a clear operation if result is not zero. Use Left/Right/Up/Down operations to put 123 in the operand. Hold button 0 down and perform Left/Right moves to get the carriage pointing to the thousands position in the result.

Now do an Add operation. The calculator multiplies the operand by 1000 from the carriage position which results in 123,000 which is then added to the result. The result was zero so now the result is 123,000. Note the counter is 1000. The operand (123) multiplied by the counter (1000) is the result (123,000).

Move the carriage to the tens position. Do a Subtract operation. The calculator multiplies the operand by 10 from the carriage position which results in 1230 which is then subtracted from the result. The result was 123,000 so now the result is 121,770. Note that the counter is now 990. Again, the operand (123) multiplied by the counter (990) is the result (121,770).

abCalcNDA

1. Installation:

To install abCalc, drag the file abCalcNDA to the Desk.Accs folder in your System folder on your boot drive. After rebooting, you should find "abCalc" in the Apple menu in any GUI application on your Apple //GS.

Alternatively, if you have the IR Finder extra installed, you can just double click abCalcNDA from the Finder whenever you want to launch it. If you reboot, you will have to double click it again to add it because it won't be loaded automatically on boot up.

2. UI Overview:

The abCalc UI is split into the following major components:

1. The list at the top is the "stack" where the numbers you are working with will be displayed. The stack always displays at least four items, even if there are not four things on the stack. Each item on the stack is prefixed with a number which is its depth on the stack where "1:" is the label for the item at the top of the stack and "2:" is the number just below the top of the stack, etc. The number at the top of the stack is displayed at the bottom of the list (did I mention that the NDA is called the "Ass-Backwards Calculator"). Any non-empty row on the stack can be selected and

you can do a copy operation on the row. The number on that row will be copied to the clipboard.

2. On the left side, directly below the stack is the entry box. This is where you can type in new numbers which go on the stack. You can actually do everything with abCalc with the keyboard. You can type in numbers or operations which manipulate the numbers on your stack. If you are typing in operations, you can type them in upper-case, lower-case or any mixture. abCalc does a case insensitive match for the operation. You can select text in the entry box and do the usual cut, copy and paste operations your selection.

3. On the right side, directly below the stack is a long list of the operations available in abCalc. The operations are sorted into an order which groups them into the following types: arithmetic, stack, trigonometry, exponentials and logical. You can scroll through the operations but be careful. Just clicking on an operation in the list will result in that operation being executed. So, items in the list operate both as a cheat sheet of the operations available and as a way to execute those operations.

4. Below the entry box is a series of buttons which make abCalc look just a bit like a classic calculator. You can use these buttons by clicking on them using your mouse or you can just type into the entry box directly. Whichever way you want to work. Note that the "+", "-", "x", "/" and "^" buttons do the same

thing as their counterparts listed in the operation list. So you can add numbers in three ways: you can click the "+" button, you can click the "+" item in the operation list or you can type + followed by enter on your keyboard. The numbers 0 to 9 and letters A to F are there to allow you to enter numbers in both decimal and hexadecimal (hex numbers consist of numbers 0-9 and letters A-F). When you click them, the number or letter is inserted into the entry box. Similarly the period and # buttons insert those characters into the entry box. See number formats for the meaning the the # character.

5. A Bit About RPN:

Let's talk about some more backward-ness. RPN stands for "Reverse Polish Notation" and it is a different way to write arithmetic expressions. People are used to things like "1 + 2" but in RPN, that would be "1 2 +". The way to think about this is "Put the number 1 on the stack, then put the number 2 on the stack, then execute the + operation which takes the last two numbers from the stack, adds them and puts the result back on the stack".

So, if you wanted to calculate "1+2" on abCalc, you would type or click the following: "1 <enter> 2 <enter> +". NOTE, you can actually avoid pressing the second <enter> if you click the + button or the + operation from the operation list. When you click a button which executes an operation or select an operation from the operation list, anything in the entry box is first pushed onto the stack. Then, it executes the operation you selected. This is just a small shortcut you can use. In my examples in this section, I will always include the unnecessary <enter>.

You can do more complex calculations by combining operations together. Imagine you wanted to calculate "(1+2)*3". In abCalc, you would type or click the following: "1 <enter> 2 <enter> + 3 *". But, what if you wanted "1+(2*3)". That is easy also: "1 <enter> 2 <enter> 3 <enter> * +".

In general, abCalc has two fundamental types of operations: unary operations and binary operations. Addition and multiplication is a binary operation because it takes two items from the stack (two - thus binary) and pushes a single result back onto the stack. A unary operation takes a single number from the stack and pushes a single result back onto the stack. An example of a unary operation is SIN which calculates the sine of a number in radians. So, to calculate "sine(4)", you would type or click the following: "4 <enter> <SIN>". To calculate "3*(sin(4-2))", you would type or click the following: "3 <enter> 4 <enter> 2 <enter> <-> <SIN> <*>". Remember, you can click SIN from the operation list or you can type "sin<enter>" into the entry box to execute the sine operation. Operations use case insensitive matching so you can enter "Sin", "sin", "SIN" or even "SiN". Whatever you like.

There are operations which are neither unary nor binary (like DROP, CLEAR and RCWS) and those are documented later.

RPN may seem unnatural and "ass-backwards" but with practice, it can start to become second nature to the point where you may dread using a standard calculator.

6. Number Formats:

abCalc operates on two types of numbers: real numbers and integer numbers. Real numbers are standard decimal numbers which may or may not have a fractional part. They may be expressed as an exponential number, like 6.283E15 which means "6.283 times 10 to the power of 15". The exponential can be negative for a very small number, like 4.712E-13 which means "4.712 times 10 to the power of minus 13". abCalc will automatically display very large or very small real numbers in exponential format.

Entering negative real numbers and negative exponentials causes a minor problem in the calculator. The "-" character normally executes the subtract operation. There are some exceptions though. If the entry box is empty, pressing the "-" character will insert a minus character into the entry box. The calculator is assuming you want to enter a negative number. If you actually wanted the subtract operation, just press "<enter>" and the calculator will perform a subtract. If you have a positive or negative real number in the entry box followed by "E" or "e", then the calculator assumes you are entering an exponential number. If you then type "-" or hit the "-" button, it will insert a minus character after the "E". This allows you to enter negative exponents. If you have a number on the stack which you want to make negative, you probably want the CHS (change sign) operation.

Integer numbers start with a "#" character. But before entering an integer, you need to know what base you are in and the bit width. By default, the calculator is in decimal mode and expects base 10 numbers. You can switch between bases by using the BIN (binary), OCT (octal), DEC (decimal) and HEX (hexadecimal) operations. The integer number you enter is interpreted using that base so if you are not sure, you may want to execute the specific base you intend to use.

After the "#" character comes a series of 0's and 1's when entering a binary number. Or numbers from 0 to 7 for an octal number. In decimal mode, you would enter digits from 0 to 9. And in hexadecimal, the digits are the numbers from 0 to 9 and letters A through F. The letters can be entered in lower or uppercase when entering a hexadecimal number. An integer on the stack has the "#" prefix but also has a suffix to tell you the current base. The suffix is "b" for binary, "o" for octal, "d" for decimal and "h" for hexadecimal. This entry and display format is often used in HP RPN calculators which abCalc somewhat mimics.

Other than the base, the other thing to be aware of with integer numbers is the current word size. By default, the calculator manipulates 32 bit integers. That means you can enter an integer from #00000000h to $FFFFFFFFh. But you can use the STWS operation to specify a different word size for your integers. If you want to work with 16 bit integers, push the real number "16" onto the stack and execute STWS. You can set the word size to any value from 1 to 32. All operations which manipulate integers respect that word size. So, if you rotate the bits in your integer to the left, then the high bit according to the current word size is rotated into the low bit. This way, if you want to do 8 bit math, 16 bit math or even 5 bit math, it is just a matter of setting your word size.

There are two shortcuts when entering integers. Regardless of the current base, you can always enter a hex number by prefixing it with a "$" character. So, you can enter the hex number 42 by entering "$42" even if you happen to be in decimal mode. Also, you can use C like syntax and enter the hex number as "0x42". Note that C syntax for octal numbers does not work. The octal number 42 in C would be represented as "042" but that cannot be distinguished from the real number 42 with a leading zero. So, these shortcuts only work for hex numbers.

Note that you can use the R2B and B2R operations to convert real numbers to integers and integer numbers to real numbers respectively.

7. Operations:

All of these operations can be entered directly into the entry box or selected from the operation list on the right side of the UI. The descriptions below are grouped into a series of related operations.

Arithmetic Operations:

+: The add operation takes two numbers from the stack and pushes the sum of those two numbers. The operation works with two real numbers and pushes a real number result. It also works with two integer numbers and pushes an integer result. And you can add a real number and an integer number. When you add a real and integer number, the real number is converted to an integer in the current word size and then those two numbers are added. The result is an integer number.

-: The subtract operation takes two numbers from the stack and pushes the difference of those two numbers. To calculate "4 - 2", you would push 4, then 2 and then do the subtract. The operation works with two real numbers and pushes a real number result. It also works with two integer numbers and pushes an integer result. And you can subtract a real number and an integer number. When you subtract a real and integer number, the real number is converted to an integer in the current word size and then those two numbers are subtracted. The result is an integer number.

*: The multiply operation takes two numbers from the stack and pushes the product of those two numbers. To calculate "4 x 2", you would push 4, then 2 and then do the multiply. The operation works with two real numbers and pushes a real number result. It also works with two integer numbers and pushes an integer result. And you can multiply a real number and an integer number. When you multiply a real and integer number, the real number is converted to an integer in the current word size and then those two numbers are multiplied. The result is an integer number.

/: The divide operation takes two numbers from the stack and pushes the ratio of those two numbers. To calculate "4 / 2", you would push 4, then 2 and then do the divide. The operation works with two real numbers and pushes a real number result. It also works with two integer numbers and pushes an integer result. And you can divide a real number and an integer number. When you divide a real and integer number, the real number is converted to an integer in the current word size and then those two numbers are divided. The result is an integer number.

CHS: The CHS operation stands for "CHange Sign". It takes a single real number from the stack and returns a real number with the opposite sign. Effectively it multiplies its argument by minus one. This operation does not work with integer numbers.

INV: The INV operation is short for "INVerse". It takes a single real number from the stack and returns a real number which is the reciprocal of that number. Effectively it calculates "1 / x" where "x" is the number it pulls from the stack. This operation does not work with integer numbers.

SQ: The SQ operation is short for "SQuare". It takes a single real number from the stack and returns a real number which is the square of that number. Effectively, it calculates "x * x" where "x" is the number it pulls from the stack. This operation does not work with integer numbers.

SQRT: The SQRT operation is short for "SQuare RooT". It takes a single real number from the stack and returns a real number which is the square root of that number. Effectively, it calculates "x ^ 0.5" where "x" is the number it pulls from the stack. This operation does not work with integer numbers.

^: The power operation takes two numbers from the stack and pushes the result. To calculate "4 ^ 2", you would push 4, then 2 and then do the power operation. The operation works with two real numbers and pushes a real number result. This operation does not work with integer numbers.

Stack Operations:

DROP: This operation just pops the item off the top of the stack. It does not matter if the value is a real number or integer number.

SWAP: This operation pops the two items off the top of the stack and pushes them back onto the stack in reverse order.

CLEAR: This operation removes all items from the stack.

Trigonometry Operations:

PI: This operation pushes the value of pi onto the stack as a real number.

SIN: This operation takes a real number from the top of the stack and calculates the sine of that number as an angle in radians and pushes the result back onto the stack as a real number. This operation does not work with integer numbers.

COS: This operation takes a real number from the top of the stack and calculates the cosine of that number as an angle in radians and pushes the result back onto the stack as a real number. This operation does not work with integer numbers.

TAN: This operation takes a real number from the top of the stack and calculates the tangent of that number as an angle in radians and pushes the result back onto the stack as a real number. This operation does not work with integer numbers.

ASIN: This operation takes a real number from the top of the stack and calculates the inverse sine of that number and pushes the result back onto the stack as an angle in radians. This operation does not work with integer numbers.

ACOS: This operation takes a real number from the top of the stack and calculates the inverse cosine of that number and pushes the result back onto the stack as an angle in radians. This operation does not work with integer numbers.

ATAN: This operation takes a real number from the top of the stack and calculates the inverse tangent of that number and pushes the result back onto the stack as an angle in radians. This operation does not work with integer numbers.

Exponential Operations:

LOG: This operation takes a real number from the top of the stack and calculates the base ten logarithm of that number and pushes the result back onto the stack. This operation does not work with integer numbers.

ALOG: This operation takes a real number from the top of the stack and calculates ten to the power of that number and pushes the result back onto the stack.

This operation is the inverse of the LOG operation. This operation does not work with integer numbers.

LN: This operation takes a real number from the top of the stack and calculates the base e logarithm of that number and pushes that result back onto the stack. This operation does not work with integer numbers.

EXP: This operation takes a real number from the top of the stack and calculates e to the power of that number and pushes that result back onto the stack. This operation is the inverse of the LN operation. This operation does not work with integer numbers.

SINH: This operation takes a real number from the top of the stack and calculates the hyperbolic sine of that number and pushes that result back onto the stack. This operation does not work with integer numbers.

COSH: This operation takes a real number from the top of the stack and calculates the hyperbolic cosine of that number and pushes that result back onto the stack. This operation does not work with integer numbers.

TANH: This operation takes a real number from the top of the stack and calculates the hyperbolic tangent of that number and pushes that result back onto the stack. This operation does not work with integer numbers.

Logical Operations:

R2B: This operation takes a real number from the stack and converts it to an integer given the current word size. The converted number is pushed onto the stack.

B2R: This operation takes a integer number from the stack and converts it to a real number. The converted number is pushed onto the stack.

AND: This operation takes two integer numbers from the top of the stack and pushes the logical and of those two numbers back onto the stack as an integer number. This operation does not work with real numbers.

OR: This operation takes two integer numbers from the top of the stack and pushes the logical or of those two numbers back onto the stack as an integer number. This operation does not work with real numbers.

XOR: This operation takes two integer numbers from the top of the stack and pushes the logical exclusive or of those two numbers back onto the stack as an integer number. This operation does not work with real numbers.

NOT: This operation takes a single integer number from the top of the stack and pushes an integer result with each bit inverted (0 to 1, 1 to 0). This operation does not work with real numbers.

SL: This operation takes a single integer number from the top of the stack and shifts each bit one position to the left, inserting a 0 bit at the low bit position. The high bit (as determined by the word size) is lost. This operation is basically like multiplying by two. This operation does not work with real numbers.

RL: This operation takes a single integer number from the top of the stack and rotates each bit one position to the left and pushes the result back onto the stack. The high bit (as determined by the word size) becomes the bit at the low bit position. This operation does not work with real numbers.

SR: This operation takes a single integer number from the top of the stack and shifts each bit one position to the right, inserting a 0 bit at the high bit position (as determined by the word size). The bit at the low bit position is lost. This operation is basically like dividing by two. This operation does not work with real numbers.

RR: This operation takes a single integer number from the top of the stack and rotates each bit one position to the right and pushes the result back onto the stack. This low bit becomes the bit at the high bit position (as determined by the word size). This operation does not work with real numbers.

ASR: This operation takes a single integer number from the top of the stack and shifts each bit one position to the right. However, the high bit (as determined by the word size) is preserved so if it was a 1, it remains a 1. This operation is basically like dividing by two where the high bit represents a sign bit. This operation does not work with real numbers.

BIN: This operation takes no values from the stack and pushes nothing onto the stack. It sets the default integer base size to binary. Any integers on the stack will be displayed in binary format after executing this operation. When entering an integer, the calculator will expect a binary number.

OCT: This operation takes no values from the stack and pushes nothing onto the stack. It sets the default integer base size to octal. Any integers on the stack will be displayed in octal format after executing this operation. When entering an integer, the calculator will expect an octal number.

DEC: This operation takes no values from the stack and pushes nothing onto the stack. It sets the default integer base size to decimal. Any integers on the stack will be displayed in decimal format after executing this operation. When entering an integer, the calculator will expect a decimal number.

HEX: This operation takes no values from the stack and pushes nothing onto the stack. It sets the default integer base size to hexadecimal. Any integers on the stack will be displayed in hexadecimal format after executing this operation. When entering an integer, the calculator will expect a hexadecimal number.

STWS: This operation takes a single real number from the stack and pushes nothing onto the stack. The real number should be between 1 and 32 and have no fractional part. The value becomes the new word size used for integers. So, if you want to do 16 bit integer math, you would push 16 onto the stack and then execute the STWS operation.

RCWS: This operation takes no values from the stack and pushes a single real number onto the stack. The real number is between 1 and 32 and is the current word size used for integers. Use the STWS operation to change the word size.

apple2048

This game requires an enhanced Apple //e or better because it uses MouseText.

Use I-J-K-M or the arrow keys to slide all tiles in a direction. Matching tiles are added together to make a new tile. On every move, one more tile is added with a random value of either 2 or 4.

Play ends when all tiles are occupied and no more moves are possible. Try to get the largest tile you can!

Key commands:

- Press escape or Q to quit at any time. Your game is saved
- Press R to start a new game.
- Press S to toggle sound on and off.
- Press H to get help.

a2sudoku

The goal is to get the numbers from 1 to 9 uniquely in each column, row and 3x3 sub-square. Move the cursor with arrow keys, I-J-K-M or mouse. Press a number key to enter a value. Press a number key while holding shift or open apple to toggle a scratch value. Use the scratch values to put in possible numbers as you work to solve the puzzle. Press 0 to clear a square. Play ends when the puzzle is solved and all squares have the correct value.

Options:

- Difficulty allows you to select easy, medium or hard puzzles.
- The "show invalid values" option puts a strike through any value you enter which is not valid because it is not unique within its column, row or sub-square. This doesn't necessarily mean the value is wrong, just that value uniqueness has been violated.
- The "show wrong values" option puts a strike through any value you enter which is not the correct value in the solution.

Key commands:

- Press escape or Q to quit at any time. Your game is saved to restart later.
- Press O to change options.
- Press R to restart the current game.
- Press N to start a new game.
- Press H to get help.
- Press U to undo your last move.

a2bejeweled

This game requires an enhanced Apple //e or better because it uses double lores graphics. A MockingBoard including speech is supported in this game but not required.

Use I-J-K-M, the arrow keys, joystick or mouse to move your selection. Hold either apple key, joystick or mouse button and move your selection to swap two jewels and match 3 or more jewels. When you match three jewels, they disappear and new jewels will drop from the top.

If you match four jewels or three jewels in two directions, then the jewel does not disappear. It will start blinking to mark it as a special gem. Match it again and it explodes taking more jewels with it. Match five jewels and a special jewel will appear. Swap it with any other jewel and all jewels of that colour will disappear.

When the score bar on the right fills, the board reloads and you level up. Play ends when no more matches can be made.

Key commands:

- Press Q or escape to quit at any time. Your game is saved so you can continue later.
- Press R to start a new game.
- Press O to select options.
- Press H to get a hint.

colourGo

You must have a colour monitor to play this game (I suppose color monitors from the United States will work too).

Your player runs to the right at all times. Press the closed apple key or joystick button one to jump. Hold the key or button down longer to jump higher. Release it to jump lower. Release and press it again to double jump.

Change the colour of your player by pressing any key (I suggest the space bar). The player will toggle between green and violet. Your player must be the same colour as any floor you touch!!! Some floors are white and because your player cannot turn white, you cannot touch white floors.

Note that if the player touches a floor from above _or_ below, you can jump again. That means you can bounce along below a floor and continually jump hitting your head against the floor above you.

The solid floor is the end of the level.

Key commands:

- Press Q or escape at any time to quit the game.