

IntCalc

version 1.0

An Integer-oriented RPN calculator (but with floating point too).

Brought to you by:

The Office of Dubious Programming

Copyright © James Preston, 1990

IntCalc is shareware. If you like it, please send \$10 to:

James Preston

1904 Miraplaza Ct. #23

Santa Clara, CA 95051

Bug reports and enhancement requests are encouraged.

IntCalc was developed using Symantec's THINK C.**Oh no, not another calculator tool!**

Yes, but this one really does have some features I haven't seen in other tools.

One problem I have with the calculators-on-computer I have seen is that they try to duplicate a calculator exactly, including all its limitations. Why put a calculator on a computer if you don't make use of the added capabilities?

I like my HP-16C, but the display is only eight digits wide. This makes it a real pain to look at a 32-bit number in binary. So the *IntCalc* display has room for all 32 digits.

Similarly, the 16C has a four element stack, but only shows me one element at a time. I find myself frequently using the roll-down and X/Y-exchange keys to see what's on the stack. So *IntCalc* shows all four elements all the time.

And since we've got all this room on the screen, why not have an optional second window onto the stack so that you can see a number in both decimal and hex at the same time? The "ALT" button brings up this window, in which you can set the base of each stack element individually.

Memory registers are great for storing intermediate results or anything you want to keep around for a while, but my own memory is not always good enough to remember what I've stored where. So *IntCalc* has a button ("RGS") that brings up a window showing the contents of all the memory registers.

Finally, I live in an ASCII world, and I occasionally want to see the hex equivalent of an ASCII character, or find out which ASCII character corresponds to a particular bit-pattern. So *IntCalc* has an ASCII mode.

(On the other hand, to show that I am not immune from the "thinking too much like a calculator" syndrome, it was only recently brought to my attention that *IntCalc* would be more Mac-like if it allowed full editing of the stack and register fields. Maybe in the next version.)

Oh, and despite the name, *IntCalc* does have floating point.

Nifty on-line help (or, what's an LDZ button?)

For those who don't read documentation, or those who just need a little memory boost once in a while, *IntCalc* provides a little direct help. Hold down the command key and click and hold on a button. A little window will popup with a short description of what that button does. (And I thought of this before I saw system 7's cute help balloons.)

Hey, why isn't this stupid thing a DA?

Because DAs are a pain to do. Originally, I assumed I'd make it a DA because that's what every other calculator was. But many people told me that with MultiFinder now so prevalent and with System 7 just around the corner (yeah, right) it would be stupid for me to spend the time making *IntCalc* a DA. And when I started looking at how much work was really involved, I decided that it would be better to get this thing out into the world and see if there was even any interest in it before tackling DA stuff. So I hereby encourage feedback on this subject. If you would use *IntCalc* more (or at all) if it was a DA, please drop me a note to that effect.

Where the heck is the "=" button on this thing?

For those not familiar with the HP-style Reverse Polish Notation (RPN), I think your best bet would be to find a friend who has an HP calculator and get a lesson, or at least borrow the instruction book. Oh, heck, I'd better say a couple of words about it here, otherwise I can't really call this "documentation".

Briefly, the working part of the calculator consists of a four element stack. The elements are labeled X, Y, Z, and T (don't ask me about that last one, I just use HP's names). When you type digits, the number goes into X. When you are done typing a number, click the Enter button. This "pushes" the stack: The contents of X move into Y, the contents of Y move into Z, the contents of Z move into T, and the contents of T go into the bit bucket. Temporarily, X still contains the entered number. If the next button you click is a digit, it will overwrite what is displayed in X. If, on the other hand, the next button you click is an operation, it will operate on what you see in X.

Clicking on a binary operator (addition, subtraction, etc.) performs that operation on X and Y in the order Y <op> X. So, for example, to compute 5-3, you would do the following:

- Click (or type) 5

- Click Enter (or type **return** or **enter**)

- Click (or type) 3

- Click (or type) -

At this point, the stack drops (the value in Z moves into Y; the value in T is copied into Z; T remains unchanged) and the result (in this case 2) is put into X. The former value in X (in this case 3) is copied into a special register called "last X". This value can be recalled by clicking on the "LSX" button.

For unary operations, such as SHL, the result replaces the previous value of X and the rest of the stack is unchanged. And again, the previous value of X is copied into the "last X" register.

For those who'd rather type than click

As expected, you can enter digits by typing. You can also enter ASCII characters this way, when the base is ASC. The **delete** key acts just like the BSP button (or vice versa, depending on your point of view), and the **return** and **enter** keys act just like the Enter button. In FLT mode, the 'e' key acts like the EEX button.

If you have an extended keyboard, the '/', '*', '-', and '+' keys on the numeric keypad will perform the associated operation. Note, however, that the same keys on the regular keyboard do not perform the operation but instead type the associated character (this is so that you can enter those characters in ASC mode). The **clear** key on the numeric keypad acts like the CLR button.

The command keys for **undo**, **cut**, **copy**, and **paste** are also enabled in *IntCalc* (see below for more information on how cutting and pasting works in *IntCalc*).

From binary to ASCII to floating point

The five bases BIN (binary), OCT (octal), DEC (decimal), HEX (hexadecimal), and ASC (ASCII) are collectively referred to as integer modes. Clicking on one of these buttons converts all elements of the stack into the corresponding base (ok, for you sticklers, it doesn't actually convert anything it just changes the base used to interpret the bit pattern). You may also individually set each stack element to any of these bases via the popup menus on the right.

FLT (floating point) mode is completely separate and may not be intermixed in the stack with the integer modes. The popup menus are disabled when in FLT mode. When you click on the FLT button, you will be prompted to click on a digit button to specify the number of digits to display to the right of the decimal point. Note that, from within FLT mode, you can change the number of displayed decimal places by again clicking the FLT button followed by the new number of places to display. This affects the display only; internally, the values are always stored in full precision.

Normally, when *IntCalc* is changed from an integer mode to FLT mode or vice versa, the stack is not cleared and the values therein are converted (as best they can be) from the old base to the new. Thus, if X contains a decimal 5, when you switch to FLT mode X will contain a floating point 5. The values are truncated when going from FLT to an integer mode, so if X contains floating point 5.6, switching to decimal will give you an integer 5.

If, on the other hand, you would rather switch modes while leaving the bit patterns the same and having them simply reinterpreted in the new mode (which is the way it works on the HP-16C), simply hold down the **option** key when pressing the new mode button. Thus, if X contains the value 123456 HEX, switching to FLT mode while holding down the **option** key will display 1.67 e-39 because that is the IEEE floating point value that results from that HEX bit pattern.

IntCalc uses 32-bit floating point because that's how big the integers are. The implementation is more convenient when everything is the same size, and it allows you to use the integer mode to decode or encode IEEE values. If there is demand, however, I could implement the larger floating point format.

Certain functions are valid in only one mode or the other. These buttons are disabled (which makes them dimmed and mostly unreadable) when they are not valid. In integer mode, the "." (decimal point), " \sqrt{x} ", " $1/x$ ", " y^x ", and "EEX" buttons are dimmed. In FLT mode, the "SHL", "SHR", "ASR", "SLn", "SRn", "ARn", "RL", "RR", "SB", "RLn", "RRn", "CB", "MSL", "MSR", "#B", and "RMD" are dimmed. The "LDZ" button is not disabled in FLT mode because you might want it for the alternate view window.

To comma or not to comma

To aid readability, *IntCalc* puts a comma between every three digits in DEC and FLT modes, and between every four digits in BIN and HEX modes. If you'd rather see all the digits scrunched together, the SEP (separators) button will toggle this display off and on.

If you like to always see all the digits, the LDZ button toggles the display of leading zeros. This applies only to BIN, OCT, and HEX modes.

Squirreling things away for the future

IntCalc has sixteen storage registers. To store the value from X into a register, click the "STO" button followed by one of the digit buttons (0 thru F). The value in X will be copied into the designated storage register. To recall a value, click the "RCL" button followed by the digit button of the desired register. The stack will be pushed, and the stored value will be entered into X.

The "RGS" button brings up a window showing the contents of the sixteen storage registers. Each will be displayed in whatever the base of X was when it was stored. The "CLEAR ALL" button in this window will reset all storage registers to zero.

Cutting and Pasting

Like all good Macintosh programs *IntCalc* allows cutting and pasting, via both the Edit menu and the command-key equivalents.

Undo (not too surprisingly) reverses the last change to the stack.

Clear resets X to zero.

Cut and **Copy** copy X to the clipboard, with **Cut** having the added effect of clearing X.

Paste copies the clipboard to X, pushing the stack first (even if the calculator was in digit entry). *IntCalc* looks at the characters to be pasted and tries to handle the pasting intelligently using the following rules:

If *IntCalc* is in floating point mode then

It will attempt to interpret the value to be pasted as a floating point value. If it is not a valid floating point value, nothing will be pasted.

If *IntCalc* is in an integer mode then

If the value to be pasted starts with "0x" then

If what follows the "0x" is a valid HEX value then

the base will be changed to HEX and what follows the "0x" will be pasted.

(This is standard C notation for indicating hex values. e.g. if the clipboard contains "0xA34F" then the hexadecimal value "A34F" will be entered into X.)

else

the base will be changed to ASC and the first four characters of the value will be pasted

(including the "0x"). (e.g. if the clipboard contains "0xBACK" then "0xBA" will be entered into X in ASC mode.)

If the value to be pasted contains all digits then

If the value is valid in the current base then

the value will be pasted in the current base.

else

the base will be changed to DEC and the value will be pasted.

else if the non-digit characters are valid HEX digits then

the base will be changed to HEX and the value will be pasted.

else

the base will be changed to ASC and the first four characters of the value will be pasted.

If the above seems a little confusing, just forget it. Most of the time when you paste something, you'll know what you're doing and it will work the way you expect it. Only when something unexpected happens should you need to refer to the above to find out what happened.

For those who occasionally make mistakes

When an invalid operation is attempted, such as the old division by zero or attempting to make a mask bigger than the word size, *IntCalc* beeps and displays a little window with an error message in it telling you what went wrong. The next time you click the mouse or type a key, this window will go away. Note that this window is not an alert or a modal dialog; if you click a button when this window is up, the function associated with the button will be performed.

If you know what the 'C' stands for in '16C', raise your hand

IntCalc duplicates the continuous memory feature by creating a file in the system folder called "IntCalc data". The following information is read from this file when you start *IntCalc* and stored into the file when you quit:

- The contents and base of each stack element.
- The base of each stack element in the alternate view window.
- The contents and base of all storage registers.
- The contents of the lastx register.
- The state of the leading zeros (LDZ button) and show separators (SEP button) flags.
- The number of displayed digits in FLT mode.
- The position on the screen of the calculator window, the registers window, and the alternate view window.
- The visibility state of the registers window and alternate view window.

You can also use the **Save** and **Open** menu items to create and restore your own data files (and, of course, you can double click on a data file from the finder to start *IntCalc* with that data).

Don't tell anyone, but I violated the interface standards

I've never liked the edict that the first click in an inactive window just activates it. Why should I have to click twice to use a button in another window? This is especially bothersome in a multi-window application like *IntCalc* wherein the concept of a single active window doesn't really apply. The main calculator window is where all the action is, and using the button or pop-ups in the other windows doesn't change that. So, my violation is two-fold: No matter which window is active, typing always goes to the *IntCalc* window. And clicking once on a button or pop-up in an inactive window not only makes that window active but also operates the button or pop-up. Note, however, that this last only applies among *IntCalc*'s own windows. If another application or DA is active, then a click on *IntCalc* will only activate it, not operate a button.

What IntCalc doesn't have

The integer modes are always two's complement. If anyone has a use for one's complement, let me know and perhaps I'll take a stab at it. There is no unsigned DEC mode.

It doesn't have the 16C's rotate-with-carry-bit functions. Does anybody actually use those?

You can't change the word size; you're stuck with 32-bit words.

It isn't programmable.

Possible future enhancements

Make it a DA.

Would it be desirable to have command-key shortcuts for all of the buttons? If so, how should they be arranged: Should I try to be mnemonic (command-A for add, command-S for subtract) which will quickly break down (there are six kinds of shifts and four kinds of rotates, but only one S key and one R key); or should I try to map the displayed buttons onto the keyboard which will be difficult due to the six rows of buttons and only four rows of keyboard keys.

Programmability would sure be nice, wouldn't it? Let me know how much you might be willing to pay for it, and I'll think about how best to implement it.

Allow normal editing of the stack fields.

A 64-bit version.

The ability to set an arbitrary word size. (Probably too difficult. This is one place where a real calculator has an advantage over a computer and a high-level language.)

This all assumes of course, that at least a few people will actually use and pay for this thing.