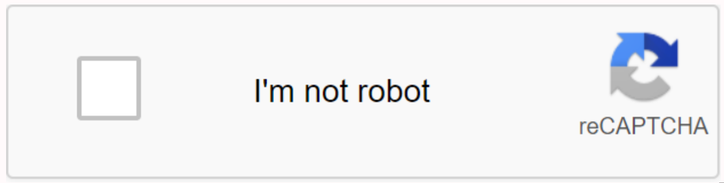

Continue

Irr ti 84 plus c

Irr (Commandment Summary command calculates the Internal Rate of Return from an investment. Commands syntax irr (CF0, CFList, [freq]) Menu Location on the small-83, press: 2nd FINANCE to access the Finance menu. 8 to select irr (, or use arrow and ENTER. On TI-83+ or higher, press: APPS get access to the Applications menu. 1 or ENTER to select Finance... 8 to select irr (, or use arrow and ENTER. Calculator Compatibility TI-83/84/+ SE Token Size 2 ur's calculator (order obtains the Internal Rate of Return of an investment, which is a measure of its efficiency. Math interpretation is the interest rate for which npv (will return 0 for the same cash flow. ur (take three discussions: an initial cash flow list (CF0), a more cash flow list (CFList), and an optional frequency list. Can Advanced Use irr(0, {list coefficient}) However, provided by a list of its coefficient: 1+.01irr(0, { undetergged list }) However, this method is limited to finding root greater than 1, and will throw an error (ERR: NO SIGNED CHG or ERR:DIVIDE BY 0) if it cannot find the following root. By reversing the list of coefficient and taking the reciprocal of the found root, you might get root less than 1, but this would still result in error if those roots don't exist either. Using Resolve (finding root in polinomals is less efficient, but more reliable, since it doesn't throw an error unless there is no root at all to be found. Solve formula for irr (requires solving a polynomial and degree equal to the total amount of cash flow. As such, there is no general formula for calculating irr (, though numeric methods are possible to find an approxsimative solution. The polynomial associated with the calculation is: (1)\start {align}\definicolor{dark}{rgb}{0.90.0.91.0.859}\pagecolor{dark}\sum_ <3>{i=0}{N}{C_i\left(1+\frac{mathrm{Irr}{100}\right)^{N-i}=0{end{align} Here, Irr is the internal percentage of return, N is the amount of cash flow, and Ct is the cash flow. For the calculator, only roots for which Irr>0 are considered to be visible. Exciting Order Requirements page 2 TI-Developing Home: 68k Welcome to TI-Developer (TI | BD), The TI-Basic Repositive Information! If you are a first-time visitor, please check out the welcome package to get you up to speed on using the site. We encourage you to become a member and participate in the community, and return often to see what changes have occurred. And above everything else, enjoy your stay! Search The Site Contains a wide range of Basic 68k content available on this site, so we recommend using the search engine or referring to the sitemap. If you can't find what you're looking for, leave a post in the forums and someone will help you. Page 3 the Lbl Commandment Summary Marks a point in the program for use with Goto and Toolbars.. Beneficiary. Command syntax: Lbl This command location menu cannot be found in any menu besides the command catalog. This Compatibility Calculator commands work on all calculators. Token Size 2 bytes (plus 1-10 bytes for label names) is the Lbl command used to mark a point in the program that orders Goto or selection from a Toolbar.. EndTBar menu can be skipped. When jumping to a label, the calculator will begin searching the program from the beginning to find the label. Limits are even imposed on a label name as on a variable name. It's okay to have a label with the same name as a variable, though, this won't cause problems. You can, theoretically, have labels in the same program with the same name, but only the first one will be used (the calculator will always find it before it finds the others). Labels do not extend beyond a program: you cannot jump to a label inside a different program or function. Also, the code inside an expr() lives in its own program as far as the labels are concerned: if you put a label in it, the main program will not get it; you cannot skip from a running routine with expr() in the main program; finally, you don't have to avoid the label names that the primary program uses. 500 - Invalid label happens when the label name uses an invalid character. branchingcompromgramvariable Page 4 68k Index Commands Some commands have a superscript next to indicating compatibility: 2.xx indicates that the command requires AMS 2.xx or higher on a TI-89, TI-92, or TI-92+ (or any 3.00+ versions, on TI-89 Titans and V200). 3.xx indicates that the command requires a Titanium or V200 calculator, as well as AMS 3.xx or higher. Flash indicates that the command requires a calculator with Flash ROM (that is, it won't work on a TI-92). Page 5 Time and Date commands version of OS 2.07 update presents several commands for dealing with times and dates. Some of these rely on the built-in clock, while others are used for formatting. TI-84+ programmers won't find many differences in function here – these commands are added in both calculator series at the same time, and are almost exactly the same. The only difference is the addition of the getTmZn() and set SetmZn() commands, with the absence of a day-between-date order. Low-Level Commands startTmr() - This command returns the actual value of a timer that is updated every second when the timer is enabled. This value does not match any current time, but can be used with checkTmr() to get a time difference. check() - checkTmr(t) is equivalent to startTmr() -t. This can be used to get the time spent since startTmr was used. Clock, ClockOff - Enable or disable the clock on the laptop. seClkOn() - Test whether the clock is enabled or not. Time commands meteTime() - Set the current time, to hours, minutes, and seconds. If the clock is enabled, this time will be updated every second time. - Returns the current time as the list {hours, minutes, seconds}. This command is not affected by time format. set SETMFmt() - Set the time format – 12 hours, or 24 hours. getTmFmt() - Returns this time format setting. getTmStr() - Returns the current time as a string, which is affected by time format (even if you can record it with an optional argument). set SetmZn() – Set the current time zone, as an offset (in minutes) from GMT. getTmZn() - Returns the current time zone. Command date meteDe() - Insert current date (year, month, and day). If the clock is enabled, this date will be set as needed. findDate() - Returns the current date as the list {year, month, day}. This command is affected by date format.meteDtFmt() - Set the date format - 1 to month/day/year, 2 to day/month/year, or 3 for years/months/days. getDtFmt() - Returns this date format setting.getDtStr() - Returns the current date as a string, affected by date format (even if you can override it with an optional argument). Time/Date Manipulation timeCnv() - Converts a number of seconds to a list of {days, hours, minutes, seconds} representing the same time span. jouOfWk() - Returns the day of week (Sunday of Saturday encoded as 1 to 7) at a specified date. Page 6 Math Calculators are built with one primary purpose: Math. Programming, game play, and everything else is high. So you'll get a number of powerful math commands. Although it may seem that they have no use of a programmer, programs sometimes need mathematical functions, and many mathematical functions can be used in smart ways. In this guide we will group the commands in the following five categories: Algebra token manipulation is the main fee factor of the 68k (TI-89, TI-92, TI-92+, and V200) calculators. With the Resolve() command, the calculator can provide precise solution to a just number of equations (of course, approximation solutions are even easier to find). Along with a dozen variations on Resolve(), there are a few commands for extracting various parts of an expression, which would be useful for writing your own algebraic tools. As on earlier calculator models, there are also logs and complex number operations, which are even better with mathematical token. Here is the complete list of algebraic commands: For basic arithmetic, 68k calculator advantages are that it can do precise calculations with integers up to 256255-1 (with roughly floating-point calculations up to 101,000-1). Here is the complete list of arithmetic commands: +, -, *, /‾, ^√(), !, ▶, 0b, 0h abs() approximative() ▶Bin ceiling() comDenom() ▶Dec E Precise() fPart() What and the ability for token calculation, the 68k calculators are much more useful for calculations than other models; they can make token differentiation and integration, as well as compute infinite sums and products. There some numeric functions as well have been carried on from earlier calculator models. Here is the complete list of calculated commands: ∫(), ∏(), ∑()avgRC()avgRC(d)open() fMax() fMin() impDif() limit 3.10() nDeriv() nDeriv()) NInt() taxi() Statistic is the one field in which the 68k calculators do not stand out compared to other SMALL models. In fact, there are considerably fewer statistical tools than, say, on the set calculators TI-83 (compare the pages on statistics). What has remaining is mostly a variety of regression models: For all of these, use the ShowStat command to display the results of a dialog box, and view the statistical system variables for more information. There are also some general-purpose commands for echantistics: Finally, you can draw data with the NewPlot command (see also PlotsOn and PlotsOff). The trigonometry the main thing to remember when doing trig is to be aware of what mode of angle you are in. By default, you are using radians, where a full circle measure 2π. The other angle modes are degrees, where a full circle is 360, and (on the newest OS versions for the TI-89 Titanium with 200 trips) credit where a full circle measures 400. The commands that currently work with these include the usual trigger functions (half of which - half of mostly useless - have been added to OS version 2.07) and the inverse, as well as commands to convert rectangular coordinates (x,y) to coordinate polar (r, θ). There are also the hyperbolic functions (there's a hyperbolic equivalent for each of the normal trigger functions). As far as math token con, you can use tExpand() and tCollect() to rearrange complicated expressions using Sin(), and expect to simplify them a bit by doing so. The whole list of trigger commands is: Page 7 Frequently Asked Questions (FAQ) This question is an attempt to answer TI-Basic questions that people ask. Many of the questions are related to each other, so it's recommended that you read from the entire list. If you have any questions that are not mentioned on the list, please post them in the forum or leave a comment at the bottom of the page. K: Is SMALL-Core easy to learn? A: Yes! SMALL-Core has the majority of the standard features and functionality that you find in other basic program variants (i.e., things like user input and variables are very similar), so if you can learn these languages, TI-Basic should be no problem. If TI-Core is your first exposure to programming, it will require some work to learn, but it is definitely worth it because TI-Core is a fun language to use. Q: How do I learn TI-Basic? A: The best way to learn TI-Basic is to download a copy of the manual and start making small, sample programs try out the TI-Basic commands. Once you feel comfortable with the commandments, you can start putting them together to create bigger Next, you should move on to learn the most advanced concepts and techniques that are part of TI-Core. Q: Where can I find information about TI-Basic? A: The Wiki you are currently on has the largest collection of SMALL-Basic information available, including commands, concept concepts, techniques, and experiences. The resource page also contains a comprehensive list of TI-Core Tutorials that you can find elsewhere on the internet. Additionally, you can download all these tutorials (and more) on the downloads page. Q: Do you have a tutorial about [topics]? A: The best way to find out is to use the search box. If you don't find what you're looking for, leave a comment in the forum and one of us will try to help you. We won't guarantee that you'll find everything about this wiki that you are looking for, since it is a constant task in progress and there are simply too many topics to cover. If you'd like to make a suggestion for a new tutorial, you can add it to the wiki-to-do list. Q: Where does the small-Basic name come from? A: Back when the language was growing in popularity and use, people wanted a simple name to refer to it that was easy to remember and told you what it was. Because it is the built-in programming language of the SMALL graphene calculator, and it is a variant of BASIC (more or less), SMALL-Basic is what is called it. You should note that the name is official, as TI has never actually given it a name (for example, try searching SMALL-Basic in the calculator manual; you won't find it). K: I've seen SMALL-Basic spell and all uppercase (TI-BASIC) and can mix (TI-Basic), but what's the correct way to spell it? A: Really, there's no correct way to spell it. It's just a personal preference. In this wiki, however, you'll probably notice that we spellED TI-Core with mixed cases. The main reason for this decision is because it's easier to read (all caps aren't very reader-friendly). Q: Which calculator supports TI-Basic? A: All the SMALL GRAPHING calculators have SMALL-Basic built-in support. Of course, the calculators each have their own TI-Basic variants (see next question). Q: What's the difference between Z80 TI-Core and 68K TI-Core? A: Simply put, a whole lot. Z80 SMALL-Core lacks all kinds of things that 68K TI-Core has, including indirections, local variables and functions, advanced picture manipulation, text in matrix, and so on. It's a shame, too, because things are very useful, and make SMALL-staples the richest in a language.K: Is there a place where I can interact with other PROGRAMMED TI-Core? A: While there is a forum available here on this wiki for TI-Core discussions and help, the best TI-Core forum in terms of user activity is U.S.-TI, which has a 68k TI-Core section. Indeed, the majority of members of this wiki are active members of Unity - TI, so you'll probably we hang around there. Game Q: Where can I find SMALL-Core games and programs to download? A: On our Resources page, you'll find several links to general TI-related sites. One of the best sites to visit for games and programs is ticalc.org, which has the largest archive of TI-Core games and the program of any site on the internet. You can also browse our own program archive, which can be found here. Q: What is an emulator? A: An emulator allows you to run a virtual form of your calculator on your computer, which is very convenient when you want to make quick changes to programs, or do any debugging or optimizations. There are several emulators available for you to use, so you should just experience to see which one you prefer. K: I downloaded an emulator for my calculator, but it won't work because it says it needs a ROM image. What is this? A: A ROM image is simply an example of your calculator, which tells the emulator that you own your calculator. It is mostly used as a protection because only one person is supposed to be using any ROM image. To download the ROM image to your computer, you just connect your calculator to your computer, and then the emulator should be able to download the ROM image off of it.K: I have an awesome idea for a game , but I don't know how to program. Can you program it for me? A: While we would like to help you program your game, we each have our own projects that we are working on and other real-world things (like school and a job) that handle our time, so we can't program your game for you. At the same time, if you have a TI-Basic program question question you need help with, we'd be happy to help you. Even better than our programming your game, though, is you programmed it yourself (see next question). K: What do I need to make games? A: The main things you need to do toys are your SMALL KAlculator and manual calculator. Before you actually implement a game, however, you should plan it out. This involves coming up with the idea for the game and working out the many details of the game: graphics, gameplay, menus, and so on. Once you have all these things figured out, you just need to put them into action. Q: What is a good tutorial for making games? A: Unfortunately, there really is no complete game tutorial available. Instead, there are several small tutorials that each cover different aspects of games. Additionally, on this wiki there are quite a few cover techniques – see for example the Special Topics section. K: Can I use a routine in this wiki in my game? A: Yes! In fact, we promote it. All the routines on this site are designed to be as optimized and efficient as possible, so that readers learn the best way to program. K: Can I use sprites from other games in my own game? A: The general consensus among the community programming calculator is using another person in your game fine, as long as you get permission to do so. However, if you do not plan on releasing your game in the community, but instead just keep it to yourself and your friends, then it doesn't really matter. Program Q: How do I draw graphics? A: 68k TI-Basic contains many graphic commands. Probably the most useful ones are the commands * Pic (such as XorPic, which can be used to display images of any size anywhere on the screen. However, other commands, such as lines are also useful. K: Can I do [work] in SMALL-Core? A: While it's possible to do almost anything in TI-Basic, if it looks beautiful and running at a decent speed is a different problem. If you've properly planned your program and made it as optimized as possible, and your program still takes a loaded minute and there's a five second lag after every key press, this is a good indicator that you should probably use assembly or C instead. At the same time, you should always strive to push the boundaries of TI-Core. Q: How do I convert a number to a string and vice versa? A: The string case() command can be used to convert any type of variable to a string. To convert to the other direction, use the handlers() command. K: My program is very big. Is there a way to manage/condense the code better? A: First of all, your program will probably shrink in size after the first time you run it, due to tokenization. If this isn't enough for you, see the optimization page for more tricks. K: Are there any characteristics without paper (Pak Egg) in TI-Basic? A: Unlike the TI-83 version of TI-Basic, virtually everything we know about the 68k calculators' TI-Basic documented somewhere. However, there are some features that the manual doesn't highlight, but are quite useful in programming TI-Basic. An example is the alternate parameter of the setMode command(). Q: How do you disable the ON key? A: That's impossible to better small-staples. Use Try. EndTry block, you can disable the ON key during text input, but the only way to disable it universal is with an assembly program or Runtime code. Q: How do I hide the code in my TI-Basic program? A: That's impossible - if someone can run your program, they can see the code as well. You really shouldn't try to hide the code too, but let others learn from it instead. Troubleshooting Q: Calculator I cannot handle expressions with several variables, (e.g. it cannot factor x^2+ to ^2+2xy). How do I fix this? A: When you type xy next to each other, the calculator does not treat it as multiplication, but as a different variable called xy. Write x* them instead and you should have no problem. K: I entered a simple expression and am confident in the result, but the calculator provides something rare! (For example, d (x^2,x) provides 10 as a result) What do you have? A: Make sure that all the variables you are manipulation in a general way (x, in the example) are actually undefined. do not, delete them using DelVar. Otherwise, the values will get replaced and you will get an unknown answer. This can result in other unusual errors as well. K: I think some of the routines on this wiki have errors in them because they didn't work for me. Could you please correct them? A: We tried to make sure that all the routines of this site work correctly and without issue. However, if you are 100% sure to enter the routine correctly in your calculator, please leave a comment on the page using the Comment function at the bottom of the page. A person will then be able to correct the routine so that it will not cause anybody else any problems. K: I played a TI-Core game and my calculator suddenly closed. When I turned it back on, my memory was erased. What happened? A: Your game had a glitch of some kind, and it caused the crash calculator. This is usually caused by Assembly Program, as the majority of TI-Basic errors are retained by the calculator. You don't have to worry so much about TI-Basic crash because they don't do any real-time damage to the calculator, but because it is very annoying to have to replace all your programs after your RAM is cleared, you should always store any relevant files in the archive. K: When I tried to run my TI-Basic program, I got this error message. What does that mean? A: Most error messages are fairly self-explanatory, but if you're still confused, you should consult our list of error messages with a more in-depth explanation. Q: The transfer program, SMALL Connect, doesn't work for me. How do I fix it? A: SMALL Connectivity issues can be an issue on both the Mac and Windows platforms. Here are some solutions listed for the Windows and Mac platforms. Mac: Uninstall and reinstall TI Connect kill process the SMALL Connect Manager X. This can be done using Activity Monitors that benefit. Start Activity Monitor in/Applications/Utilities, then find SMALL Connect Manager X in the list of processes (if you can't find it, just type tick on your keyboard). Select it, then click the X icon at the top left corner of Activity monitor. Click Force Kite. Then you can restart the device manager, and it should detect your calculator. In some cases, disrupting your calculator or turning it off will always cause SMALL Connect Manager X crashes, and you'll need to kill it every time you need to connect it to your computer. To make this process easier, there is a script here that you can use this process's automatorization. Use a different USB port. Windows: Reinstallation should fix more problems. Check to make sure that the plug (on both ends) are firm in the port. Assembly / C Q: How is TI-Basic compared to Assembly or C? A: TI-Core is easier to learn with the program, but it is rather slow because it is an interpreted language. TI-Basic has many good graphic commands, they're still slower than they are or Program C; also, SMALL-Basic PROGRAMS are limited in control over the calculator. K: Is it possible to convert SMALL-Core assembly? A: No, it's not. There is currently no working program available that will convert SMALL-Core assembly (Note: I say working because people have tried to create SMALL-Core in Assembly converter, but no one has completed one yet), so the only way you can convert a TI-Basic assembly program is to not learn Assembly and bring the program yourself. You could also try asking an Assembly programmed to bring it for you, but most people won't do so unless the program is very small. However, if you are looking for faster your TI-Basic program, you have the option to use BasicBuilder 3.0, which packages your TI-Basic program to an application. Q: I want to use an Assembly and My TI-Basic program, but I can't know how to use it. Can you help me? A: Unfortunately, we really can't do much for you. What we recommend is that you contact the author of the Assembly program and ask them for help. They wrote the program, so naturally, they should be able to answer any questions that you have. [[/div]] Page 8 Click here to edit the contents of this page. Click the Active editing of individual sections of the page (if possible). See Title for a link edit when available. Paste content without editing the whole page source. Check out how this page has evolved in the past. If you want to discuss what's on this page – this is the easiest way to do it. View and manage file attachments for this page. A few useful tools to manage this site. See the linking page and include this page. Change the name (also URL address, maybe the category) of the page. See the source wiki for this page without editing. View/insert parent page (used to create bread and structured layout). Administrators notify if there are content objections to this page. Something doesn't work as expected? Find out what you can do. Documents Wikidot.com section help. Wikidot.com Terms of Service – what you can, what you should not etc. Wikidot.com Privacy Policy. Page 9 Strings and the Commands A code are a collection of letters (usually called characters) in order. They are most commonly used to display text, although they can be adapted to any purpose. See the page character codes for a table of the characters that can be part of a string. To write a string, use quote marks around the characters you want it to contain. For example, Hello is a string containing the characters H, e, read, read, and o, in the following order. Using the string() command, you can also create a string from any other type of variable. On the TI-68k calculators, strings have an advantage over lists or matrixes – they are a random access structure, which means that accessing the end of the string is just as fast as accessing the end of the (for a list, on the other hand, the calculator has to go through all previous elements in the list found in an element at the end). Although the strings are left accessed normally, so shortened strings are at a disadvantage compared to lists, long enough strings will beat out lists and matrixes, at least for accessing a specific character. Except for the constraints of memory, there is no limit to the amount of characters in a string. There are two special operators that can be used on strings – (connection) and #(indirection). Operator to connect, && join two strings simultaneously – Hello> world returns Hello World. The indirect operator, #, replaces a string that contains the name of a variable by the variable itself – #f(x) is the same as f(x) (this can be very powerful, especially for non-algebraic variables such as pictures). (annoying, there's no easy way to tap #, on a TI-89. You can select it from the character menu (2nd CHAR, 3, 3) or from the catalog) the relationshipal operators (=, ≠, >, ≥, <, ≤) can also apply to strings. It's easy to see how = and ≠ work: the two strings are equal if they are made up of the same characters. For other relationship operators, they use alphabetical order: a string is less than another if it would come before it's in a dictionary. there you go.. not quite – there is a problem with uppercase and lowercase letters. These comparisons are actually done by comparing the character codes of each character in the strings, so all lowercase letters are considered after all uppercase letters. That is, cat < dogs, as you would expect (because cats come before dogs in the dictionary), but Chen < cats. The hard aspect of strings is accessed through a character. To do this, you have to use the middle command() middle(string, start, start, length) will return character length from string, starting from Start. As a special case, middle (string, n.1) will return the nth character of string. The count characters start from 1. Of course, there are several commands specifically designed to be used with strings (though not as much as we would like). All of them except Change() and Rotate() are found in the string menu (Press 2nd MATH to access the popup math menu, then select D: Channel). char()expr() format() inString() left() left() medium() right() Right() Clean() Switch() Page 10 Code Timings This document the speed of certain commands in 68k TI-Basic. Now obviously, the times provided here (in seconds) will vary from model to model and even from calculator to calculator (due to battery level, free memory, and other factors). However, something that hasn't changed is the relative speed of the commands. So if you come here to see if then. Else.. Fennif is faster than when(), the information will be useful about any calculator. Elsewhere in this guide, you might see statements such as Footer() is faster than bar() any reason or evidence. The information on this page is the reason with evidence behind them. Test Format for each test done on this page, a variation of this function was used: tempting(n) : Func: Local t, i : startTmr()→:P You,1,n:End :<command>Approx(1000 { checkTmr(t), 1 }/n) : EndFunc this input function, n, us, are the number of the loop times of the order. The bigger this is, the more accurate the test; however, it will take more time. The output of this function is a list like {185 5}. The first is the measure of the time: how many milliseconds have been measured to <command>work. The second of these is an error margin for this measure. The error occurs because startTmr() and checkTmr() use whole numbers in seconds, so the current time might be cut by just under one second in either direction. In the tables below, such a result will be written as 185± milliseconds. For commands that take a very short amount of time, this alternate form can be useful. It takes into account the time that the To.. That's why Buckle himself takes on the run. Tempted: Func: S Local, t, i : startTmr()→:P Y I,1,n : End : checkTmr(s) → :startTmr()→t:For i,1,n<command> ;:EndFor:approx(1000{checkTmr(t)-s,2}/n) : EndFunc contributes your own test Feel free to experiment with code timings, and to put your results up in this page. However, make sure you use the same format and method! Also, list the calculator model and the OS version (found in the F1, A: About... menu). Unless stated otherwise, all tests on this page were done with a 200 trip calculator and OS version 3.10. That's it for details and explanations. Now come the real timings! Loops with Conditional Air() command is often a simpler alternative to if.. Else.. End block, when only a single value needs to be stored. It's smaller; This test attempts to ascend against if it is faster as well. When() the code tested was If test code was : If condition then : 1: Else: 0: End both tests made and a true as well as a strong condition. Time code(ms) when(), true condition 8.2±0.05 when(), false condition 8.15±0.05 If.. Else.. Phenif, true condition 8.5±0.05 If.. Else.. Phenific, false condition 8.65±0.05 Conclusion: When() contains order, but not a lot (though there is a difference that cannot be explained by error alone). Furthermore, the true results and false conditions' conditions are so close, they are probably equal. Variables although the OS() and # (indirect) commands do things differently, they overlap considerably. If you have a string with a variable name inside, both # and expr() can be used to get you the value of that variable. (In case you're curious, the difference is that #, unlike expr(), can be used to refer to the variable itself, not only its value – but expires() can be used for the value of an entire expression stored in a string). Both commands take 2</command></command></command>when tokenized, so there is no difference in size. The question is – which is faster? Time string (ms) # 14.35±0.05 expr() 17.05±00.05 Conclusion: It's better to use # than expr() when you can. List vs String vs This matrix tries to compare the time it takes access to the list element, strings, and matrix. Before the test, all three variables were initialized to comparable values: The list was set to Element 2048, all of them 0 (as an integer). The string was set to Space 2048. The matrix was set to row 32 in 64 columns (a total of 2048 elements), all of them 0 (as an integer). The elements in each can have one byte of data. However, only stores the entire string with no commands, and only a single header; as a result, the list with the major matrix variables. It is known that while the internally stored strings as an array, listing and matrix are stored as expressions in Polish reverse notation. As a result, it is expected that accessing any element in the string will take an equal amount of time; access to a component in a list or matrix will take more time as the element gets further from the beginning. Although this has not been tested, the total size of the variable should not change the access time significantly. List String Matrix Time String(ms) Time String(ms) Time(ms) List[1] List[1] 9.1±0.05 Middle (str,1,1) 40.5±0.1 Match [1,1] 9 .8±00.05 list[500] 36.8±0.2 middle (str,500.1) 41.8±0.1 mat [1.64] 13.35± <0>100.05 list[1000] 64.5±0.5 middle (str,1000,1) 44.3±0.1 martha[32.1] 115± list [2 1228] 122±1 Middle (str,2048.1) 49.6±0.1 Match [32.64] 118±1 Conclusion: Initially, of course, it is faster accessed through lists or matrix elements than string characters. However, the price increases fairly stew: for around 600 components, the string will have. Of course, for storing numeric data, the list will have an additional advantage, but in the length run (say, for 800-1000 elements, depending on the method), the string will be better even for that. You might also consider using multiple small lists rather than a large one if it is possible. As for the matrix, it's clear that the time price increases faster going down a column, than across a row. This can be a factor in how you choose your data oriented when it doesn't really matter. You probably won't improve the total time, but the appropriate orientation will make sure that the weather doesn't vary dramatically. <<Program Overcome Schools >> This Page 11 Routines Page is a short routine collection that can be used as subroutines in larger programs. By contrast, the Program Samples page links to all games (which may use some of these routines). Technical examples have yet a third type of program – short programs that can be of some value by themselves, but they are presented here mainly to demonstrate the use of some programming techniques in an actual program. When routine contributors, please Such guidelines: A routine should be self-contained, and relatively short of length. A routine should be thoroughly tested to make sure it works correctly. Set inc: Routine as the starting point for pages. When the routine page ends, link to it from here. Page 12 Sprites Sprites is a common name for a small image displayed on the screen (usually, this image is moved around as well, or repeated several times, such as in a tilemap). In SMALL-68k Basic, support for built-in sprites: A picture file can be any size (up to the total size of the screen) and can be displayed from anywhere using commands such as RclPic. Sprites are always rectangular; even if the image itself is a circle, the sprite data will include empty pixels around the circle to make it a square. Sprite Logic Contains four basic methods of displaying a black-and-white sprite. The simplest one is to overwrite wherever this is before. Alternatively, you can apply AND, OR, or XOR logic to each pixel in the background with the sprit to produce the result.Overwrite RplcPic uses overwrite logic to display a sprit. This is often useful when you intend to cover the whole screen with sprites, for example. The drawback is that this can delete other trace pieces on the screen.XOR XorPic uses XOR logic to display a sprit. This means that for each pixel black in the sprite, the pixel is reversed on the screen. Logic XOR is the simplest logic to use for sprites, since the same XORing sprite in the same location a second time will erase it and restore any pixels under it. However, if sprites are going to overlap frequently (as opposed to occasionally), XOR Sprites might not look that big. OR RclPic uses OR logic to display a sprit. With OR logic, pixels can be set, but never cleared. For many possibly-overplaping screen sprites, plot sprites and OR look better. However, they are not as easily deleted (you might store the background before drawing the sprite, but this can lead to very complicated code). AK andPic used with logic to display a Sprit. With logic, pixels can be cleared, but never set, so it has limited use of itself. It can be used as an equivalent of OR logic when drawing in white-on-black instead of black-on-white. Also, and logic is useful in making sprites (see below). Masking Masking is an advanced technique that allows sprites to be transparent in some places and optical to others. To do this, you use two sprites: the sprite itself, and an auxiliary spirit, called a 'mask', which contains black pixels everywhere you want the result to be transparent, and otherwise light pixels. To show the sprite masquerading, you first AND sprite to the auxiliary, then OR or XOR (either one task) the main sprite. Like any sprite technique except XOR, masked Sprit destroys what was in the background below them, so if you ever want to delete the sprit, you have problems. solution: if you have one sprite moving around the screen while the background remains the same, you can store the background of a picture size in the screen. Then, whenever the sprite needs to be erased, you RplcPic the background, request the sprit. Another solution: for several sprites, you can stopic the area below the sprite of a picture, then RplcPic just which part of the background when you want to delete the sprite. This allows these sprites to be deleted independently, but is complicated to manage (the simplest way is to use indirction). Tilemaping cards are a widely used technique to map to games like RPGs. The idea is to use small (such as 8x8) sprites in a grid template to cover the entire screen. Every type of plot (e.g. grass, water, brick walls) has its own sprit associated with it. A similar idea can be converted into side-scrolling games like Mario. The easiest way to store a tile map is to a matrix of integers; converting from integers to these sprites easily, sprites names are corresponding 'tile1', 'tile2', ... and so on, or something similar (then #(tile&string(n)) will find you the nth sprit). The title card can be displayed in two nests For.. Endening loop: For row,1, height: For col,1, width: RplcPic #(tile&string(map[row,col])), 8* row-8,8 * col-8: End:End:End (this code conceded 8x8 sprites, a tilemap is stored in the 'map', and sprites stored in the 'tile1', 'tile2', etc.) a more efficient method use char() instead of string() - see the order page() command page for more information. Scrolling rather than displaying each screen as it happens, it will look better if the map can be scrolling. This is possible using a combination of Stopic and RplcPic. Use StoPic to store all but one row or column in the screen of a temporary picture. Then RplcPic joins the same picture, but one row or column is over. Finally, use a To loop to replace the remaining row or column with headings. Here's a left roll example (fix0 and col0 variables are used to follow your location of the map): : StoPic temple,0,0,8 * height,8* (width-1) : RplcPic temple,0,8: col00-1→col0: For row,0,0,height-1:RplcPic #(tile&string(four [rows+rows,col0])), 8*row,0: End an even more advanced technique is scrolling list. Here, the same idea is applied, except you shift the screen one pixels over at a time. Also, since you can't draw negative coordinates in the screen, you might have to apply some difficulties when scrolling left or up, or just don't use the whole of the screen. Optimizing XorPic this technique is useful when using XorPic to display a moving sprit, especially when it can only be moved in a few directions. The basic idea is that the XOR operation on bits is associated: (screen XOR pic1) XOR pic2 = screen XOR (pic1 XOR pic2). The typical form using XorPic to display a Sprite moving underneath XorPic pic1, row, © efraze old row +1→ © coordinates XorPic pic1, rows, © columns mapped to this new optimization location, creates a new picture that is the combination of pic1 and pic1 changes down row 1, combined using XorPic. You may think of this picture as the pixels that need to be changed in order to move pic1 down 1 row. If this picture is stored in pic2, then the optimized code is like bar: XorPic pic2, range, © update range sprite + 1→ © the row coordinate asset in this technique is showing one sprite is twice as quickly as showing two. On the other hand, it requires that there are two pictures to appear. Also, the code can get complicated if more than one direction is possible. Storing pictures A large, graphic-intense TI-Basic game might be finished using many photo variables for sprites. Although you might release these photos with your program, this might get annoying since the user has to transfer all these photos to the calculator. A better way is to use an installation program that creates these photos. There are three methods: A Picture Reference This Method involves only one picture file containing all the sprites next to each other. The installation program reminds this photo of the screen, then uses StoPic to store sprit to separate variables. The advantage of this method is that it's small and fast; but it's not as fun aesthetics as it might be because the user has to see the reference picture during installation. NewPic This Method uses NewPic to store each sprite individually. This method avoids drawing in the screen, and is still fast. However, a drawing is that NewPic uses a matrix of store sprites, which has two components per black pixel in the sprite. That makes even small sprites take up a lot of space in the installation program. Coded Newpic for this fix, you might try to encode the sprites in a different way (an example is using a large integer – a single integer can store up to 254 bytes of data, which is good for about 2032 pixels, more than enough for any lead). Then the installation program created the matrix for NewPic on the fly. This fixed the only drawback to NewPic, the size. However, another disinvantage appears: creating the matrix is a slow process. <<Dialogs Animations >> List 13 and the commands A list is a collection of elements in order. On the TI-68k calculators, lists can include any mixture of any valid variable type in an expression: you can have lists of numbers, list strings, list of values of true or expressions. You can even mix and match type variables – it's perfectly all right there is a string in one element, with a number in the next. The only special case is listing: These are types allowed, but they are called matrix, and have some more restrictions. Lists written using curly brackets { and } around the elements, by commas. For example, {1.2.3.4.5} is a list containing the digits 1 through 5 of the following order. You can access a certain element from the list by writing the number of elements to the element you want in [] brackets after reading: list[5] would select the 5th element in lisvar. The count elements begin with 1. On calculator pattern earlier, listing had the random access property: Access any element in a list took the same amount of time. This was possible because the lists were enforced in numbers. On the 68k calculators, since lists can mix types of elements, they have no random access: the calculator has to go through the entire list found in an element, so the larger an index is, the longer it takes access to. This is not significant for short listings. But take the 100th element, for example, taking roughly twice as long as taking the 1st element, and the time keeps increasing lineyely, so it can very slow to access the latest elements in a long list. Except for the constraints of free memory, and at the time it takes access to elements, there is no limit on the number of elements a list can contain. Operations on many lists of commands, including mathematical commands and others, can be extended to list by applying them to each element in the list. An example is Sin(): sin({1.2.3}) { Sin(1) Sin(2) Sin(3) } If a command has more than one argument, there are two ways to extend it to apply lists. One is to use it with a list with a regular argument: Then the command will be applied to each element in the paired list with the regular argument. Here is this way illustrates with Mode() mode ({10.20.30}.7) {3 6 2} The other way is to make both arguments list. In this case, the lists must be the same size, and each element in the first list will be paired with the corresponding two elements of the second. For example: Mode({10.20.30}, {7.8.9}) {34 3} although in the following example, Mode() might be extended in both ways, sometimes only one is possible. PtOn, for example, (as well as other point orders) can be used with two numbers, or two lists, but not with a list and a number. round(), on the other hand, can be used with two numbers, or a list with a number, but they will provide a sense expression when applied to two lists. A remarkable special case is the basic math operators (+, -, *, /‾, and ^), which can all be used with lists (in both ways). List-Specific commands of course, there are commands specifically designed for use with lists. Several of these, such as Dim() or Introduction(), can also be used with strings. Many of the following commands are found in the list menu (Press 2nd MATH to access the popup math menu, then select 3:List). increment() cross() cumSum() Dim() dotP() exploit▶list() Fill left() Δlist() List▶mat() Match ▶List()max()min() NewList() polyEval() Product Right() Introduction() seq() Change sum() Multiple Statistics commands can be applied list as well. Conditional conditional statements that contain conditional statements, such as If, When(), and Meanwhile, accept list of values to true as well as single true values. The check will be interpreted as true if and only if each element in the list is true, effectively combining each element in the list with And. The most common way to list the true values to be created is with relationshipal operators (=, ≠, >, ≥, <, ≤) applied to list. The single value will be 'false' unless both sides of the relationship are equal size lists, in case the lists will be compared elements by elements, returning a list. Page 14 Glossary Arguments One of the values that an act of commands on. Array A collection of numeric variables is arranged in a list or matrix for processing by the calculator. Each element in an array is referenced by its index – position of the array. American Standards Code ASCII information. It uses 1 byte (8 bits) to hold a character, and thus is limited to 256 different

characters. The calculator uses a modified version of ASCII interpreters to store characters (see character code), but the interpreter deals TI-Basic and token instead. Assembly of other built-in language programming calculators are SMALL GRAFING, apart from SMALL-Basic. It is much more difficult to learn with the program, but at the same time it is much more powerful, and can be used to create highly sophisticated games and programs. You can also compile C in an assembly program. The two-digit binary system (bit) number is based on 0 and 1, similar to the normal decimal system route based on the numbers 0-9. For example the number 10101 to binary represents 1*24+0*23+1*22+0*2+1, or 21 to decimal. To enter a number of binary on a TI-68k calculator, write 0b followed by the number. Bit A binary digit (0 or 1). Branch a departure from the performance sequence of program statements. An unconditional branch causes the calculator to go to a specified location in the program each time the branch statements meet. A conditional branch transfer control is based on the result of some logical requirements. Breaks a point in a program where the program's execution stops, used when debugging the program. In SMALL-Basic, the following breaks can be created with the Text command. Bug An Error in a Program (see Also Debugging). Byte A string of eight bit. The calculator uses a single byte of information to encode the letter A, as well as most token. Bytes (or kilobytes – a kilobyte is 1024 bytes) are used to measure the amount in memory of a program or variable taken up on the calculator. C a programming language is popular on many platforms. Using TIGCC, Program C can be compiled for TI-68k calculators. Character A letter, number, punctuation symbol, or special graphic symbol. They are encoded in internal ASCII format, but the TI-Basic interpreter instead. Clock time unit cycle in a CPU. It is so small that even the most primitive assembly instructions take multiple clock cycles to run; the amount of clock cycles it takes for a single TI-Basic order can be in the millions. Order a word or token that says the calculator to do something. Example: RclPic, To, max(), etc. They are sometimes subdivised to functions with Concatenation instructions Join two or more strings together to make a string longer. Associated - is the connect operator. For example, TI-&gtDebaz= TI-Basic. Conditions of a logical expression that the calculator can evaluate to be true or false. Constant A specific number, list, matrix, or string that is provided as a fixed value of the program (such as 9.024 or Hello), rather than contained in a variable. CPU (Central Processing Unit) the brain of the calculator, where the calculator controls the process and execution of everything. Cursor A small, flash square showing where a typed character is displayed. A cursor (in the form of a cruise) is also used on the graph screen to input a point from the user. Frequent numeric information in form. When data is used by a calculator program, it can be hardcoded or softcoded. Debugging Fixing a program to remove pugs, or errors. There are many techniques for debugging, some universal, some specific to program calculators. Decimal A base system 10 numbers using the 0-9 symbols. It is the default number system used by the calculator. Default a standard feature or value that is supposed to be calculator without specifications. For example, the RclPic displays a picture of the coordinator (0.0) by default (the top left corner). This can be replaced with additional arguments to specify the coordinator to display in. A message box appears on the screen for input and output. Execute another name to run a program or command. Outlining a number indicating the power that contains a number or expression is to be raised, usually written to the right and above the number. For example, 26 = 2x2x2x2x2x2x2. In scientific connotation, the power of the ten that mantissa is multiplyed by calculating the actual number. Flash memory to ROM / archive on the TI-68k calculators where applications are stored. Floating point format is used in TI-Basic for storing decimal. He uses scientific notation to show all numbers. Flow Chart A diagram in geometric shape linked by arrows showing the progression of a program. Flow charts are handy for developing complicated programs and illustrations how programs work. The Word function has several meanings in the context of TI-Basic PROGRAM. It can mean a type of order that returns a value - such as gcd() or sin(). Or it can mean a user-defined program type that returns a value. Any sort of visual display on the screen, such as graphs, templates, and designs, both stationary and animated. Hardcode Hardcoded data is used by the program that results from the logic in its commands. It's very easy for the program to read, but may be impossible to change. Frame materials are circuit and other electronic parts that make up a calculator. Hexadecimal A system number 16 base using symbols 16, 0-9 and A-F. It is used as a convenient sorthand way to express binary code, because every four bits of binary contain a corresponding hexadecimal symbol. To enter a number of hexadecimal on a TI-68k calculator, write 0h followed by the number. Index A position of an array. In SMALL - Basic, the index array starts with 1 (so that the first element in a list is counted 1, the second 2, and so on). In some other languages, the index starts with 0. Entry means not which data is entered into the calculator by the user, mostly in the calculator key (though the link port is also a form of input). Instructions In the case of TI-Basic PROGRAMMING, this is often used to indicate a type of order that does not return a value, but stands alone: for example, line, or Enter. It is also sometimes used to describe commands in general. Integer, positive or negative (or 0). Usually in TI-Basic, a number is an integer relative by default, which means all its digits are stored exactly. You can also use floating point numbers for approximating and decimal calculations. Interprets the algorithm stored inside the calculator that runs a TI-Basic program by implementing the appropriate system routines. Iteration One rehearsal in a loop. For certain commands, the calculator has an internal loop for which the total number of iterations is relevant, and affects accuracy. The Keypad Panel of Keys used to enter programs and data in the calculator. Lists a variable sequence that can be accessed and modified by their position. In TI-Basic, lists can contain any mixture of type variables. Loop A group of one or more consecutive program lines is designed repeatedly, either a specific number of times, or until a condition is met. Mantissa's portion of basic numeric in an express number of scientific notation. At 3,264E+4, the mantisa is 3,264. Matrix A programmatic two-dimensional variables that can be accessed and modified by their position. In TI-Basic, lists can contain any mixture of type variables. Memory of the two locations is different (RAM and ROM) where calculator programs and data are stored. All variables and programs take part in memory to store, which makes the size of a program or variable very important. MHz the CPU calculator speed runs in. It ranges from 10MHz to 16MHz on the TI-68k calculators. It measures in millions of clock cycles per second. One-way system number to write a number. Two common systems are used by decimal ones (Arabic) systems, and numerical system. The calculator uses the binary system internally, but the decimal system is what it usually shows. The symbol operator is used in calculations (numeric operators) or in relation comparison (related operations). The math operators are +, -, *, /, ^. The relationshipal operators are >, <, =, ≥, ≤, ≠. The logical operators are and, or, xor, do not optimize the process of improving the run time, length, or memory usage of a program. Information from the calculator, such as graphics on the screen. Also, means by which data leave the calculator, in either the link cable, graph link, or USB cable. OS (Operating System) The internal program that runs the calculator and includes all the fonculity necessary to use the calculator. The program lists instructions that tell the calculator what to do a job. Programming someone who writes programs. Numerical language programming or alphabetical commands that the calculator can understand, and execute. RAM (Random Access Memory) A temporary memory, meaning one of which data is stored so long is electrical power is applied. Data in RAM can be accessed or changed and lost if the pile is removed from the calculator. Rehearsal of a program that calls itself as a subroutin (or a function that is defined in terms of terms of itself). See rehearsals. ROM (Read only memory) Certain instructions for the calculator are forever stored in ROM and can be accessed but cannot be changed. Data in ROM is not lost if batteries are removed. Scientific Notation A method of expressing very large or very large numbers using a base number (mantisa) times ten rises from some power (outlining). Scroll Shifting part of the screen provides the illusion of view only part of an image larger than the screen.Softcode Softcoded data is the information stored in an array at the beginning of the program, and reads from it later. It is usually slower than hardcoded data, but the advantage is that the same code can often be used for different data stored in the same way. Sprite a small image which is moved around the screen using code and/or repeat multiple times on the screen. Usually synonymous with order. String A variable type that stores text as a range of characters. Subprogram A program segment that can be used more than once during the execution of a program, such as a complex set of calculations or a print routine. You can make subprograms by using the Define command. SMALL (Tectonic Musical Instruments) the musicians in the SMALL grafing calculator. In addition to graphics calculators, they also make a great asortment to other electronic devices. TI-68k The official name for the SMALL calculators with a 68k processor: the TI-89, TI-89 Titanium, TI-92, TI-92 Plus, and Travel 200. these are calculator described in this section of TI|BD.TI – Basic the official name of the programming language found on TI CALCULator. Token One instruction or symbol to tokenize TI-Basic which takes 1 up to 3 bytes and is used to encode TI-Basic programs. User when you are talking about the effect of a program or order, the person runs the program or orders about his calculator. Variable A name is given to a value that can vary during program execution. A variable is a memory location where values can be replaced by new values during program execution. Page 15 Dialog Although input on the I/O screen (with commands such as Insert and InputStr) may be useful for some programs, dialog boxes are usually more efficient. A dialog box is a typical message box that appears on top of wherever your screen is currently on, but without replacing anything (so the background is restored when the dialog box is exited). Case basics a dialog can be created by setting a dialog.. Enddlog blocks the sender of a program. The dialog will be displayed when this part of the program is run; after the exit dialog, the program will resume after the EndDlog. Just dialog: EndDlog would provide an empty dialog box, and in fact cause an error. To provide content for the dialog box, use the following commands: Title gives the dialog a title. There may be at most a title. If there is no title, the top of the dialog box will be blank. Text add a line of text into the dialog. The text must be a string, so use the string() to order to display any other data types. Request Add a text entry box to the dialog. Requests take two arguments, a text line and a variable to store them. DropDown Adds a dropdown menu to the dialog. DropDown takes three arguments: a line of text, a list of strings for the dropdown, and a variable to store them. Text and Request can be used by themselves, providing a dialog box with no other elements in it. Advanced details at the bottom of a dialog box, there are two buttons: ENTER =OK AND ESC=CANCEL. If the ESC key is used to get from a dialog, the dialog is true cancelled: the variables that would have been affected by Requests and DropDowns are left as they were (usually, undefined). Fortunately, there's a way to check if this happened. The variable system ok to keep track of the way that the last dialog came out. It does so in a somewhat unusual way: if ENTER has been pressed, its value is the floating-point value 1.0, and if ESC has been pressed, its value is 0.0 (perhaps this is a hold of calculator TI-83 series, storing all true values in this way). Ok the variable is not affected by a Text Instruction outside dialog.. EndDlog (which can also be exited using ESC, but not has two buttons), nor by a PopUp menu. It works when a Request instruction is used by itself outside dialogs.. Enddlog. Request Alpha-Lock still stores a string of but it is meant to be used for all kinds of input. As a result, there is an option to turn off alpha-lock in alpha-lock for it, if you're going to be entering a number. To do so, add a third argument to Request evaluated 0. This was boldly only added with OS version 2.07, which could be an issue. It also doesn't actually do anything about the width calculators (since these have a QWERTY keyboard and don't need alpha-lock). However, it's still there on wide calculator for compatibility, and just doesn't affect anything. Unfortunately, if there are several requests in a single dialog.. Enddlog blocks, they cannot have different alpha-lock settings. Either they all have alpha-lock or none of them do (the setting for the first request order is used, and the rest are ignored). Compatibility If the text of one of the elements in a dialog is too long to fit in one line, the dialog provides a dimension error. However, the length is enabled to vary text from the TI-89 and TI-89 titans to the other, screen calculator. So a dialog that works on a calculator can give an error on another. You might think that more text fillings on a screen calculator, but that's actually not the case, because they use a different font for everything except headings. The current limitations are given in this table: Widescreen commands (TI-92/+/V200) Standard (TI-89 / Titanium) Title 50 approx chars. 39 Text chars (Dialog) 38 chars approx. 39 text chars (simple) 34 chars approx. 37 chars Request 20 chars 20 chars DropDown 35 chars. 35 chars Note: Approx. means that due to the variable width of a font used, the actual upper limit varies. For DropDown, the upper limit is about the total length of the description and longer options. It makes sense to write dialog boxes and text that adjust both sets of conditions at the same time. Then there is no risk in the Perverted Dialogue ever crashing! Default values If the variables being stored in by Request or DropDown are not undefined, they are used as default values: For requests, if the variable value is a string, it's pre-entered and highlighted in the request box. For DropDown, if the variable value is a number, the corresponding option is the initially selected one. This can be useful for changing menu options. Alternatively, you can use it for instructions or default custom values, for example: Input reply here→only ask Ans:, respond if the variable has the wrong type, it doesn't show up, as it has indulged. These Snippets dialogs are some common code spies that involve dialogue, which might come in handy in almost any program. Of course, the text can be customized to your needs. Confirmation dialog :D ialog: Title are you sure? : Unsaved Data Text will be lost! : EndDlog: If ok = 1 Then : © Continue: Else: © abot: End Load / Save File : Main→fold :D g: Title Load Record, fold: Variable Name Request, var : EndDlog : If ok = 0:© Abort: # (fold&\)→file © to save © file, instead of doing: file→ #(fold>&&var) It is a good idea to ask for the folder with variable names separately, because it clears up questions about syntax. Also, if you move: main→ to an earlier part of the program, a folder has been loaded from once will be a default value in the dialog. In fact, for this purpose, you might want to keep two variables for saving and loading files, to allow for different default values. Error dialog: Try: © main code: Else: © If an error occurred: © handle expected error, then: Dialog: Title Error : Text An unpanted error occurred. : Error Code Text: &string(error) : Continuing Text and Program? : EndDlog : If ok = 1 Then : ClrErr :Goto Start : Else : © Clean Vary : Passerr : EndTry : EndTry A dialog box as this will ensure that a bug in your code is not as a bad result as it would otherwise – a professional approach to error handling. You might want to add something such as Please mail with a bug report in <email address=>the message. Replace PassErr with something like ClrErr: Goto left if you don't want to let the user see the error message, under any circumstances. <assembly=overview=sprites=>> Page 16 Click here to edit the contents of this page. Click the Active editing of individual sections of the page (if possible). See Title for a link edit when available. Paste content without editing the whole page source. Check out how this page has evolved in the past. If you want to discuss what's on this page – this is the easiest way to do it. View and manage file attachments for this page. A few useful tools to manage this site. See the linking page and include this page. Change the name (also URL address, maybe the category) of the page. See the source wiki for this page without editing. View/insert parent page (used to create bread and structured layout). Administrators notify if there are content objections to this page. Something doesn't work as expected? Find out what you can do. Documents Wikidot.com section help. Wikidot.com Terms of Service – what you can, what you should not etc. Wikidot.com Privacy Policy. Page 17 Error This tab lists all the possible error messages that may occur. The error number to the left is not displayed, but is stored in the system error variable, which can be used in a Try.. Andtry Block. Some error messages are very explicit (such as #810), but others are less so. Where the error message itself gave too little information, the following table explains the error more thoroutly. A big part of the information here was taken to Appendix B in the TI-89 manual. Error Number Description 10 A</email> function did not return a value 20 A test did not resolve to TRUE or FALSE this error usuallywhen you compare an undefined variable, in a statement such as If.30 Argument cannot be a folder name 40 Argument Error 50 Arguments mismatch two of the arguments to be of the same type. For example, PtOn can be used with two numbers, or two lists. But a number and a list cannot be used simultaneously. 60 Arguments must be a Boolean expression or relative integer for use with logical commands such as or They can be applied to two values true, or bitwise to two integers. The 70 argument must be a decimal number 80 argument must be a label name 90 Argument must be a list of 100 arguments must be a March 110 argument must be a Pic 120 argument must be a Pic or string happening with a title used in a toolbar. Pic the part only applies to the wide calculator, by the 89 or 89 Titan Titan 130 argument must be a string 140 argument must be a name this variable can also indicate a variable variable name such as 1xy. 150 arguments must be an empty folder name cannot be deleted if the empty .160 argument must be an expression for example, zero (2x + 3 = 0,x) is not valid because 2x + 3 = 0 is an equation. Use zero (2x + 3,x) instead - a = 0 is implicit with this command.161 ASAP or Exec string too long this error caused by the program ram assembly limit. The limitation is not present on HW1 Calculator; on others, it's 8k and AMS2.03 or lower, with 24k and AMS2.04 or higher. If you try to run an assembly program beyond the limit, you will get the following error .163 attribute (8-digit number) of objects (8-digit numbers) by getting 165 batteries too low for sending/receiving product codes 170 bound to lower to be less than the upper limit. This error can occur, for example, and to get the zero on the graph. 180 The Break in on key has been pressed during a calculation or while running a program. This error usually cannot be taken by Try.. EndTry block, unless it occurs at a text prompt. 185 check error 190 Circular definition circular definition in a variable to hold: for example, a + 1→a (if that's a definition). But circular definitions of a function are handled by a limitation on profession rehearsals. 200 Invalid Expression Constraints See the page for | (with) operator for more details. 210 Data type An argument is of the wrong data type. 220 Dependents limit the same variable cannot be used as both an integration variable and a bind. For example, ∫ (Sin (x), x.0,x) should not be enabled. 225 Different Eq setup 230 Dimensions An index out of the limits of a list or matrix. For example, list[5] when the list is equal to {1.2.3.4}. 240 Dimensional mismatch Some commands require the lists or matrix arguments to match in size. For example, you cannot add the {1.2} lists and {1.2.3}. 250 Divided by Zero Take That, Jason Anderson! 260 Error Domains Some commands only accept numbers in a certain range. For example, anse() only works with numbers 1-99. 270 Duplicate variable name 280 Other things and invalid outside of Si.. Then block Properly, actually, Try. EndTry Block also uses Else. But you have not heard me say so. The 290 EndTry missing the Match 295 statement Too much erased occurs when a solve item runs for too long without finding a solution. This usually means no one, except in case truly terrible. 300 Expect 2 or list of 3 elements or this matrix happening with commands dealing with 2 or 3-dimensional vectors. 307 Flash Application Extensions (functions or programs) by finding 308 flash applications not getting 310 First arguments of nSolve must be a universal English translation expression: the only undefined variable that can be in inserted() in expression is the one you are resolved for. 320 The first argument of solving or cSolv must be an equation or inequality For example, solving (2x + 3,x) is invalid because 2x + 3 is not an equation. Of course, zero() has exactly the opposite issue. 330 Records sometimes the like calculator was cyconic. This error happens in the VAR-LINK menu if a variable is stored in a folder that does not exist. 335 Graph Function y1 (x).. y99(x) is not available in Derived Diff Derived 345 Units Inconsistent 350 from Range 360 String Indirect is not a valid name 380 Invalid Anse() is what, that many responses have not been stored yet. 390 Invalid Scoring 400 Invalid value 405 Invalid axis 410 Invalid command 420 Invalid folder name 430 Inv Help for the current mode settings 440 Invalid involves multiplying the syntax a(b) is only used for function calling, not for multiplying a and b. If a function is not a defined function, suppose they are calculators you have tried to multiply, and it provides this error .450 Invalid in a function or current expression A user-defined function cannot change global variables, or use certain commands. This also happens in the prompt, such as the Data Editor/Matrix. 460 Invalid in Custom.. EndCustm blocks 470 Invalid in dialog.. Enddlog Blocks 480 Invalid In Toolbar.. EndTBar Blocks 490 Invalid in Try.. EndTry Block 500 Invalid label name labels have the same limitations as variable names. 510 Invalid List or Matrix List can only be 1-dimensional (list) or 2-dimensional (smooth list). Lists are matrix, and anything that is not rectangular (e.g. {1, {2.3}}) is not allowed. 520 Invalid Outdoor Custom.. EndCustm or Toolbar.. EndTBar Block 530 Invalid Outside dialog.. EndDlog, Custom.. EndCustom, or Toolbar.. EndTBar Blocks 540 Invalid Outside dialog.. EndDlog Blocks 550 Invalid Outdoor Function or Program 560 Invalid Outdoor Loop.. EndLoop, To.. End, or While.. EndWhile block 570 Invalid pathname 580 Invalid polar complex You might think this is a ban against military bases on Paul North. Actually, this is used with the Command ∠. For example, (i∠2) is invalid. 590 Invalid syntax blocks for the miscellaneous errors and dialogs.. EndDlog, Custom.. EndCustm, and Block. 600 Invalid Table 605 Invalid used in Unit 610 Invalid variable names in a local declaration reserved variable, for example, cannot be local. 620 Invalid variables or function name variable names to be used for a built-in function. 630 Invalid variable reference 640 Invalid vector syntax 650 Link transmission 665 matrix per diagonalizable 670 or 673 Memory 680 Mi missing (690 missing) 700 missing 710 missing] 720 Missing } 730 Missing Start or End Block Syntax For example, If without an End to go with it.740 Miss then to Si.. FenIf blocked 750 No is not a function or program 765 No function select 780 No solution to find Most commands usually think of something smart to return, instead. So this error only occurs with the interactive graphics tools. 790 non-algebraic variables in 800 expressions that are not real result only happens if the calculator is in actual number mode. 810 Not enough memory to save current variables. Please delete variables that are not required on the Var-Link screen and re-open editor as current OR re-open editor and use F1 8 to clear editor. 830 Slip the possible range of a floating point number is between -101000 and 101000 (not inclusive). However, sometimes an overflow is replaced by infinite instead. 840 Plot setup 850 Program does not get 860 Repetition limited to 255 deep calls 870 Reserved names or variable system variable systems cannot be deleted, for example. 875 ROM - Resident routine not available 880 Sequence setup 885 Error Signature 890 Singular Matrix 895 field needs one selected function and is used for 1st-order equation only 900 Statistics. 910 Syntax An expression or entry just doesn't make sense. For example, 2+2+. 930 Too few arguments 940 Too many arguments 950 Too many subscription too much is more than two. 955 Too Many Variable Variable 960 Idle Good to 965 Without Software License or Flash Applications 970 Variable or Flash Application in Use 980 Variable Lock, Protected, or Archive 990 Variable name limited to 8 characters 1000 Window domain variable 1010 Zoom 1020 Internal Error 1030 Protect violation page 18 68 SMALL-Core for 83 SMALL-Core Programmers This tutorial means as an introduction of 68k TI-Core for programming that is already fairly experienced with BASIC SERIES CORE PROGRAMMING. Instead of re-teaching many things, this tutorial highlights the differences between the two languages. Large FeatureS A large calculator of the 68k calculators is the ability to perform symbolic calculations. This contains many applications: you can deal with expressions such as x2 + 2 * x + 1 but treat x as an unknown, or deal with the exact value of √π/3 without approximative reading and floating-point values. The calculator does not have some statistical commands, but it has much more power to calculate commands (it can make token derivatives, integral, and finished with psalm, among other things). This has no immediate program application, although you can often find an unexpected use of these commands. A very programming-relevant difference, on the other hand, is the advantage of tougher errors. Using the Try and EndTry statements, your program can identify whether an error occurred, and maybe even recover from this (or at least display a custom error message). Also one of the highlights is how many photos have been imposed. They now can be any size and must appear with any logic. True Real Time multiplayer game is now possible with the SendCalc command, something that was impossible on the 83. The most specific differences described below tend to combine to make programs run faster, and allow for a program style closer to programming a serious language on a computer. Commands on the TI-89, commands can be entered in letters by letter, and must not be selected in a menu. In practice, programs and functions are understood, perform a set of orders taking 1 to 3 bytes of a program. Many commands have been added or removed between the two languages (see the command index for a full list of commands). In addition, the following commands have changed in spelling: There are two more changes in general. First, the names of many commands have been truncated where they were longer than 8 characters: This is the maximum for a command name on the TI-89. An example is RclPic, equivalent to 68k of RecallPic. Second, use the brackets after an order now following a strict convention. Instructions - commands that do not return a value - do not require parenthesis (e.g. If, Text, etc.) Functions - commands that return a value - require brackets (e.g. sin(), meteMode(), etc.) Same function with no arguments using brackets (e.g. getKey(), startTmr(), etc.) Many commands have been added. However, as far as statistics go, the 68k calculators are inferior, even, in the TI-83 range; The most functionality is currently limited to regressions, and the calculator doesn't even know internally how to calculate most distribution probability. The way variables are stored have supported major changes to the TI-83 series. All variables now share a common naming system: The name of a variable can be up to 8 long letters. Variables can also be set to different files, which may not nest but otherwise are very similar to file folders on a computer. By default, variables are stored in the 'main' folder. On the surface, the same variable types were on the TI-83 calculator: you have numeric variables, lists, matrixes, strings, photo variables, equations, database graphs, and a new, named data. These are a little different, however. To begin with, numeric variables can now either floating-point (similar to on the TI-83 series) or Integer (which do not have a decimal place, but there are several hundred digits of precision). As a consequence of the token operations on a 68k calculator, you also have expressions: formulas that are only evaluated as much as possible. The lists are very different. They now can hold any combination of numbers, expressions, and character strings. That makes them more powerful, but also slows them down significantly. The data is basically matrix, but with the new capabilities (and speed limits) of the new lists. Matrix stayed up though, limited to numbers, but kept their pace. You can effectively emulate an old list by using a matrix with only 1 row/column, Boolean variables will also cause some confusion for scheduled TI-83: Instead of boolean (true/false) operations returning a one or a zero that can be used in the same manner as regular integers (1, 3, -5, 4.8, etc.), boolean variables now all have a class of their own. This means that you cannot use a simple if (variable) statement to check whether there is a variable yet being defined, and cannot use shortcuts in expressions involving a bracture statement to determine whether a certain term in the expression is used (e.x. (k>2) (x+1) + 3x). Instead, you'll have to use the sivar() command to check if there is a variable that has been defined and returns to regular If-then block for conditional compliance. Programs are also regarded as variables, on the same level as any other: you can even define a program in another program. They imitate built-in commands, and can even provide parameters. Using the locale, you can declare local variables that reset their old values once a program finishes running. You can also define functions, similar to programs but return a value. Functions have some other limitations, though: they can only use local variables, and cannot modify any global aspects of the calculator (so graphic commands, for example, are limited to programs). With local variables, and with the ability to define functions and programs, you can program in a procedure language style. Instead of inserting the whole code into the program in one block, you can split it up to functions with subprograms that are defined at the beginning of the main program. The whole issue of memory color (caused on the TI-83 range by jumping from cord blocks and goto) is not present in 68k TI-Basic. Loops has offsets that are linked with the end of the start, so the program does not need to keep a stack to be aware of what to be done with End instructions. There is no longer memory cost entering a loop (or any other type of code block), so it's impossible to hit this memory the way. Optimization Most trivial optimization types from TI-83 series are invalid on the 68k calculators. For example, closing brackets, quotes, and brackets are now required – but add any size to the program, since it's tokenized and converted to postfix notation. The variable's anse does not play an important role: even if the anse() command exists to replace it, it's not modified by storing variables inside a program, so it's mostly useful on the home screen. A large part of optimizing 68k gun around attention usage in listings. List of variables no longer have random access: Access to the last element in a list is slower than having access to the first element. For that reason, going from a list to a Loop is about the worst thing you could do. Graphics 68k gives the programmed more options and graphic displays. Sprites, which have to be displayed using various tricks or libraries on a TI-83 series calculator, are built in 68k as picture variables. These photo variables support little similarity to the functionality D-like they have on a SMALL-83+. They can be any size from 1x1 to the entire screen, and can be stored from and reminded in any part of the graph screen. There are even several commands to display a sprit using different logic. Apart from these very powerful commands, more ordinary commands were also puffed up. Practically all graphics commands have a point with a pixels equivalent, so you're free to choose one or the other to use (usually, you'll want pixels). The command circle now draws instantly circles, as opposed to taking several seconds. Instead of being forced to choose between your home screen and graph screen, the choice is between graph screen and Program I/O screen on the 68k calculators. Program I/O screen is a separate home screen for programs, which is limited to text (but the text must not be aligned). In addition, both screens can be spiced up using dialogs, which mimic the appearance of a popup window on a computer, and are great for data to import without having to delete anything from the screen. Another major addition to graphics set in order is the newly created dialog feature that's newly created. New commands add extra I/O capabilities that do not interfere with the I/O program or the graph screen, allowing for in-program data. There is a limit of graphics on the 68k calculators – they cannot be drawn on the top menu bar and bottom status bar, so they are effectively limited to only 2/3 in the screen. Assembly libraries can be used to access the entire screen, but this impossible in SMALL-Basic alone. Close this Word page provides an VISION of some of the 68k features, but it's not, and it cannot, be complete. There are other pages you might visit to get a better picture of 68k programs: Command Index FAQ Program Sample However, the best way to try to learn the language is to experience first hand with it. Page 19 Clear Table Code contains the values returned by getKey() for each (key, modified) pair. Unlike the getKey found on TI-83 series 68k kalkilatris yo 'getKey () retounen kle ak modifye. Sou TI-92 a, TI-92 +, ak Vwayaj 200, layout a kle se diferan. Efe a menm (egzanp F8) toujou koresponn ak vale a menm (egzanp 275), men se jwenn diferan (pa peze F8, nan egzanp sa a, olye ke 2nd + F3). Se kle a alfa ranplase pa bouton an gen tan pwan pou kle yo flech. Avi ke kod kle yo pou tape yon karakte koresponn ak sa karakte karakte kod. Pou egzanp, kod la kle pou sezisman A se 65, ak ord (A) ap retounen tou 65. Sa a menm aplike nan kod yo kle pou karakte entenasyonal yo ki pa montre nan tablo sa a. Key Modifier None ⇧ (shift) 2nd ♦ (diamond) alpha Result Value Result Value Result Value Result Value Result Value F1 F1 268 F1 268 F6 273 Y= 8460 F1 268 F2 F2 269 F2 269 F7 274 WINDOW 8461 F2 269 F3 F3 270 F3 270 F8 275 GRAPH 8462 F3 270 F4 F4 271 F4 271 F4 271 TblSet 8463 F4 271 F5 F5 272 F5 272 F5 272 TABLE 8464 F5 272 ♦ (diamond) copy 24576 cut 12288 alpha a-lock ESC ESC 264 ESC 264 QUIT 4360 PASTE 8456 ESC 264 APPS APPS 265 APPS 265 SWITCH 4361 8457 APPS 265 ON ON 267 OFF HOME HOME 277 HOME 277 CUST 4373 HOME 277 HOME 277 MODE MODE 266 MODE 266 ▶ 18 _ 95 MODE 266 CATALOG CATLG 278 CATLG 278 i 151 ∞ 190 CATLG 278 ← (backspace) ← 257 ← 257 INS 4353 DEL 8449 ← 257 CLEAR CLEAR 263 CLEAR 263 CLEAR 263 8455 CLEAR 263 X x 120 X 88 ln (4184 e ^ (8280 x 120 Y Y 121 Y 89 peche (4185 sin^{-1} (8281 y 121 Z Z z 122 Z 9 (4186 cos^{-1}(8282 z 122 T T 116 T 84 tan (4180 tan^{-1}(8276 t 1166 ^ 94 ^ 94 ^ 94 π 140 θ 136 ^ 94 | (avek) 124 F 70 ° 176 FOMA 8316 f 102 ((40 B 66 { 123 b 98) 41 C 67 } 10 125 © 169 c 99 , 44 D 68 [91 8236 d 100 ÷ / 47 E 69] 93! 33 e 101 × * 42 J 74 √ (4138 & 38 j 106 - 45 O 79 VAR-LINK 4141 kontra + o 111 + 43 U85 CHAR 4139 kontra - u 117 ANTRE 13 ENTER 13 13 ANTRE 4109 APPROX 8205 ANTRE 13 STO▶ → 258 P 80 RCL 4354 @ 64 p 112 = 611 Yon 615 ' 39 ≠ 157 yon 97 EE 149 K75 ∠ 159 S 8341 k 107 (-) - 173 ESPAS 32 ans (1) 4372 8365 ESPAS 32 . 46 W 87 > 62 ≥ 158 w 19 0 48 V8 86 < ≤ 15 v 118 1 1 49 Q 81 34 8241 q 113 2 2 2 50 R 82 \ 92 8242 r 114 3 3 51 S 83 INITE 4147 8243 s 15 4 4 2 76 6 : 58 8244 l 108 5 5 53 M 77 MATH 4149 8245 m 109 6 6 54 N 78 MEM 4150 8246 n 110 7 7 55 G 71 ∫(4151 8247 g 103 8 8 56 H 72 d(4152 8248 h 104 9 9 57 I 73 ; 59 8249 i 105 Arrow key Modifier None ⇧ (shift) 2nd ♦ (diamond) alpha/@ ⇐ (left) 337 16721 4433 8529 33105 ⇑ (up) 338 16722 4434 8530 33106 ⇒ (right) 340 16724 4436 8532 33108 ⇓ (down) 344 16728 4440 8536 33112 ⇖ (left+up) 339 16723 4435 8531 33107 ⇗ (right+up) 342 16726 4438 8534 33110 ⇙ (left+down) 345 16729 4441 8537 33113 ⇘ (right+down) 348 16732 4444 8540 33116 An easy way to find the value of a keypress without having to consult this page is to write a short program to output key codes : : Local k: 0→k : While = 0 : getKey()→k: EndWhile: Text String(k) : EndPrgm Page 20 Characters This code contains the character codes used internally by the calculator. The following table can be useful in particular with the char() and ord() commands. Characters in these ranges can be used for variable names: 48... 57 (number, as long as they do not start a variable name) 65 ... 90, 97... 122 (the usual alphabet, case sensitive) 128 ... 139, 141... 148, 181 (Greek letter, case sensitive: ω is not the same as Ω) 192 ... 214, 216... 246, 248... 255 (international characters, case sensitive) Note that the value 2 (STX) cannot be used on the home screen, because for some reason it does not exist in large font. Value char Value Char value Char Value Char Value Char Value Char Char 0. NUL 32. space 64. @ 96. ' 128. α 160. ... 192.C 224. to 1. SOH 33. ! 65. A 97. a 129. β 161. ¡ 193. Á 225. á 2. STX 34. 66. B 98. b 130. Γ 162. ¢ 194. Â 226. â 3. ETQ 35. # 67. C 99. c 131. γ 163. £195. Ã 227. ã 4. EOT 36. $68. D 100. d 132. Δ 164. ¤ 196. Ä 228. ä 5. ENQ 37. % 69. E 101. and 133. δ 165. ¥ 197. Å 229. å 6. ACK 38. & 70. F 102. f 134. ε 166. ¦ 198. Æ 230. æ 7. NICE 39. '71.G 103. g 135. ζ 167. § 199. 231. 8. BS 40. (72.H 104.h 136.θ 168.√ 200. ER 232. 9. TABLE 41.) 73. I'm 105. i 137. λ 169. © 201. And 233. 10. LF 42. * 74. J. T. 106. j. 138. ξ 170. a 202. Ê 234. ê 11. VT 43. +75. 107. 139. ∏ 171. « 203. Ë 235. ë 12. FF 44. , 76. He's 108. He's 140. π 172. ¬ 204. Ì 236. ì 13. CR 45. - 77. I'm 109. I'm 141. ρ 173.¯205. Í 237. í 14. (closed) 46. . 78. N 110. n 142. ∑ 174. ® 206. Î 238. î 15. ✓ 47. / 79. Oh 111. o 143. σ 175. - 207. ITE 239. They are 16. ■ 48. 0 80. P 112. p 1444. τ 176. 208. Ð 240. ð 17. ◀ 49. 1 81. Q 113. q 145. φ 177. ± 209. Ñ 241. ñ 18. ▶ 50. 2 82. R 114. r 146. ψ 178. ² 210. J. 242. 19. ▲ 51. 3 83. S 115. s 147. Ω 179. ³ 211. Ó 243. 20. ▼ 52. 4 84. T 116. t 148. ω 180.$^{-1}$ 212. Ô 244. ô 21. ← 53. 5 85. U-117. u 149. And 181. μ 213. Õ 245. õ 22. → 54. 6 86. V 118. v 150. and 182. ¶ 214. Ö 246. ö 23. ↑ 55. 7 87. You are 119. W 151. i 183. · 215. × 247. ÷ 24. ↓ 56 88. X 120. x 152. r 184. +216. Ø 248. ø 25. ◀ 57. 9 89. They are 121. They are 153. T 185. ¹ 217. Ù 249. ù 26. ▶ 58. : 90. Z 122. z 154. $\low {x}$186. o 218. Ú 250. ú 27. ↑ 59. ; 91. [123. { 155.$\bar {y}$187. » 219. Û 251. û 28. ∪ 60. <92. \ 124. | 156. ≤ 188. d 220. Ü 252. Mue 29. ∩ 61. = 93.] 125. } 157. ≠ 189. ∫ 221. Ý 253. ý 30. ⊂ 62. > 94. ^ 126. ~158. ≥ 190 ∞. Þ 254. þ 31. ∈ 63. ? 95. _ 127. ♦ 159. ∠ 191. ¿ 223. ß 255. ÿ Page 21 68k Basic Starts Welcome Kit at The SMALL-Developer Developer (TI| BD) 68k Basic Start Pit! This tutorial is designed to help the new programmed 68k get their feet off the ground. The tutorial is of chapters that each has their own focus, and is meant to be read in sequenty order. If you have questions or get stuck, leave a post on the forums and someone will help you. Happy coding! Page 22 Of Your Imagination Use is using a program for the first time. You have no prior knowledge of the program; someone just put the program on your calculator without giving you any instructions and now you are trying to figure out how to use it. After literally pressing all these buttons on the calculator and trying all kinds of key combinations, you give up and tell your friend the program was useless. You then delete the program because you calculate it's worth noting if you can't use it. This example is not based on any particular program (I don't want names, but most importantly speak in the general sense), but it's reasoning with many program users who have had a similar experience. What this problem really is about is poor user-friendship – most commonly known as usage. The definition of usage is simply how easy it is for people to use a program, and how much value it provides. While ilities can take on many different forms, there are some essential things that you can do to do a more user-friendly program. In game Help Probably the easiest way to make a user-friendly program is not including some help in games. While you ideally want your program to be so easy to use that a user can simply pick it and figure out how to play it, not every game so right, and the average user probably needs some help. The best place to include help in a program is as one of the options in the program menu. When the user comes across the menu, they'll see the help option and can select it to see the help. The help doesn't need to cover every detail minute of the program, but rather just explain the purpose of the game and detail what keys are used to control. :P opup { Option 1, Options 2, Help }, option... : If option = 3 Then : Dialog : Title Help: Target text in game is ... : Text Use the ENTER key ... : EndDlog : EndIf : EndIf because most people don't like using help unless they have, you should try to limit your help to one or two screens at most. At the same time, if you have a very complex game with all kinds of features and many of the keys need to operate it, then it would be appropriate to include help for all things. The general guideline is that the amount of help needed correlates to the size of the game. Protecting the next user things you can do to a user-friendly program is to protect the user by themselves. Often times in a program you'll want to think about what might go wrong and try to either prevent it from happening or tell the user of the wrong. Preventing it from happening involves you, the programming, programming of security protection for the user that they don't even know that something goes wrong. Says the program calls for the typed user in a number between 1-1000 and the user types of 5000. If your program just goes on with that value, it will probably crash at some point later on. Instead, it is necessary to check the value and ask for the number again if it's wrong. For example: Enter a number 1 to 1000,n: While n 1000: Dip Number to be 1 1 1000! : Enter the number again, n : EndWhile Sometimes it might be impractical to check if a<1 or =n=>input is valid. In this case, an alternative is to use Try.. Else.. Fentry block. If an error occurs, the program will jump to else part of the block, and the error code is stored in error in the system error variable. You can either display a generic error message or try to use the error code to figure out what went wrong. Another key teacher in making a user-friendly program is to include useful features. Since the audience targets frequently in high school, one feature to be appreciated is a key teacher. This is a special key that the user can use to quickly exit the program. When the teacher comes around, they then want to be able to return to the home screen] so they don't get their calculator taken away. This problem is quite easy to prevent with a teacher key. In each program there is a main loop that runs throughout the lifetime of the program. You need to add a check for whatever teacher key you want in the place of the main loop to which you check for user input. Make sure, of course, what the user knows which key is the teacher key! : Loop : .... : getKey()→key: if key =<teacher key=> : Exit: ... Troubleshooting the size and number of cards, this can take a while. If the user doesn't know what's going on, they probably will think the program is stalled or something else goes wrong. While there are a couple different ways you can cut down on the loading times for maps (see subprograms and compression), the easiest way to solve the problem is by simply telling the user what's going on and showing the user some progress. You don't need to do anything fancy (in fact, you probably wouldn't be because that would just waste valuable memory), just something helps the user understand the situation. For example, say you're randomly inserting mining throughout the card (it's a Mineseper game), you could then you just show a Mines Ranking message on the screen and then, there is a loop for the progress indicator which match the current map: For x,1,20: © fill the card with min : Output 0,0, Ranking min: &string(x)>/20 : End of the Last Principle at the last important point of user-friendly is to follow the KISS principle. For individuals</teacher></1> who have not heard ofit's an acronym that stands out to keep it simple stupid. The basic point of KISS is to not clusters your program with unnecessary features and useless flu. It also insights makes the program easy to evaluate for those who don't have access to a recall. It's not uncommon to see a TI-Basic math program (i.e. quadratic solve) that has a menu, on screen and credit scrolling, and includes some games in case you somehow get bored to solve quadratic equations. While things aren't themselves bad, they are completely not suitable in a mathematical program. There is a certain elegance that comes with programs that do one thing and that does it well. This is known as a Unix_philosophy, and should really be what every program strives for. <<<Facilities && Cleansing Overload School >> Page 23 Basic Program Assembly are executed by calling built-in functions already written for you on the calculator. Basic programs really aren't executed, they are interpreted by the calculator's operating system. This means that the calculator 'read the programs, and then execute the appropriate function. For each order that reads to the calculator, several native commands can be executed. This means that the calculators of the Central Processing Unit will have to do more process to execute the program. This is usually not a problem for small, simple programs; but if you are considering making a game, or other complex application, it is probably a good idea to use Assembly as your programming language.

How Assembly Programs execute Assembly programs executed naturally. This means that the Central Processing Unit is able to directly interpret your code, instead of having to rely on commands from the operating system. This also gives you more control over the calculator. Too much control? Before sending any program assembly to your calculator, make sure it is from an audit source. This is because when you execute an Assembly Program, you provide the comprehensive program to control of the calculator. An assembly program can directly modify both RAM, and ROM, enabling it to delete the operating system, installing a virus, your log. This is an Assembly Program is a Native Application, meaning it is directly executed by the Central Process Unit. Should I use Assembly? If you are creating a rather simple program, or function, then you should probably stick with Small-Basic. And if you are creating a more complex program, such as a game, then you might want to consider Assembly. Because assembly runs faster, your game will have better performance. Where can I start? Two main types of assembly programs are either written directly in Assembly, or in a compiled language, such as C. A compiled language translated into Assembly, making it easier to learn. However, programming directly in assembly gives you more control. I am a beginner in assembly If you are a beginner, and never write in C, or C++, then you should start out with C.C is easier to write, and run just about as fast. Starting with the list of tutorials below: Techno Plaza Tutorial Autids ChesS Tutorials General C Tutorials I have scheduled in C, or C++ before I am completely familiar with the concepts of programming, and have experienced in an at least one medium-low language language. This does not include C#, VB.NET, or JAVA, as these are considered high-level languages. Start with the list of below tutorial: <<Saving Data Overtime Dialogs >> Page 24 System Variable Systems to special reserved variable names used by some of the internal commands. They exist outside the folder structure which the other variables are located in, so that they can be accessed in the same way from any folder. Unlike normal variables, which take between 1 and 10 reference bytes, system variables always take 2 bytes, no matter how long the Name.Graph variables are derived variables contain the derived that get graphs for each graphic mode. They include: y1(x)–y99(x) in function mode xt1(t)–xt99(t) and yt1(t)–yt99(t) in parametric mode r1(θ)–r99(θ) in polar mode u1(n)–u99(n) and ui 1–ui99 in sequential bite z1(x,y)–z99(x,y) in 3D mode y1'(t)–y99'(t) and yi1–yi99 in differential equation mode Cursor Variables The cursor variables include xc, yc, zc, tc, rc, θc and nc. Some of them are updated every time the cruiser is moved around on the graph screen, and are especially useful with the Enter command. However, the rules that determine which ones to get updated are a little difficult: xc and yc are still updated, regardless of any settings. The rest get updated depends on graphing mode: zc for 3D mode, tc for parameteric modes, rc and θc for polar modes, and nc for sequence modes. Additionally, rc and θc get updated even outside polar graph mode, if the graph format is set to coordinate polar. Windows Variables variables define the parameters in the graphene window – are not only used for graphics, but also with point commands such as PtOn. The most fundamental of them is xmin, xmax, ymin, and ymax: these determine the lower and upper limits of the window. There are also more advanced settings: xscl and yscl determine the distance between tick marks on the axis, if the axes are enabled. Δx Δy determine the distance between two pixels next to each other. They are calculated automatically from xmin-ymax, but you can set them yourself (xmax and ymax will be adjusted to fit). xfact and yfact determine the factor by which the window is stretched when you zoom in or zoom out. Some windows variables are specific in graphene mode: In function mode: xres determines the number of pixels between sample points for graphs (a higher value means lower quality). In parameter mode: tmin and tmax determined in the variable when graphene. tstep determines the inscription of the variable variable between two sample points on the graph (a higher value means lower quality). In polar mode: θmin and θmax determine the set of the θ when graphening. θstep determines the invoice of θ variables between two sample points on the graph (a higher value means lower quality). In sequence mode: nmin and nmax determine the values we evaluate to: u(nmin), u (nmin + 1), ..., u (nmax) will be evaluated. plotStrt and Trace determine our values that actually get graphs: starts at plotStrt, and increases by tracing every time. In 3D mode: zmin and zmax (similar to xmin and xmax) control the upper and lower limits of the grafing pane, for the coordinates of z.zskl (similar to xscl) to control the distance between tick marks on the z axis, if the axis are allowed. zfact (similar to xfact) controls the factor by which the z-stretch coordinator is pulled in or out. xgrid and ygrid determine the resolution of the filframe grid. eyeθ, eyeφ, and eyeψ control the viewing angle (eyeθ is the angle with the x-axis, eyeφ is the angle with z-eyeψ axis is a rotation around the resulting line of eyes) oncountry is the number of contour and graphs. In differential equation mode: t0 determines the t-value for the initial requirements. tplot and tmax determine the variable range when graphic. tstep determines the inscription of the variable variable between two sample points on the graph (a higher value means lower quality). return determines the number of curve solutions mapped if you do not provide an initial condition. diftol (with the Runge-Kutta method) and Estep (with the Euler method) determine a step size for calculation. fldres determines the number of columns for the slope field, if one draws. time determines the point in time where a directional field is mapped (if only one draws at all). fldpic is a photo variable that stores the slope garden to avoid renewing it if it's unnecessary. Zoom graph Many of the above window variables have a variable counterpart zoom, prefix and a z. These are saved by the ZoomSto commands and ZoomRcl. These variable statistics are created when you calculate a curve to fit a data set, using one of the following commands: LinReg, MedMed, Quadreg, KibikReg, Kwatreg, PowerReg, Expreg, LnReg, or Logistics. regeq(x) is the curve that was calculated, as a function of x. Recognized is a list of the calculated coefficient. core is the correlation coefficient (a measure of direction and goodness in fit) of a linear model. R2 is the square of cor, but can be calculated for all models. A value closed at 1 indicates a good fit; a close value of 0 is unfortunate. medx1, medx2, medx3, medy1, medy2, and medy3 are calculated by the MedMed method. These Example statistics variables are calculated by OneVar's and/or commands (only those that deal with a single variable are calculated by OneVar). $\low {x}$ and $\bar {y}$ is the averages in each data range. Σx and Σy are the sums. Σx2 and Σy2 are the sums of the squares. Σxy the sum of the products in matching pairs in the two data sets. minx, maxx, miny, and maxY is the minimum and maximum. Sx and Sy are the standard deviations. σx and σy are the standard deviation populations. nstat is the number of elements in a data set. medStat, q1, and q3 (for OneVar only) are the median, first quell, and third quown. The other variables the rest of the variable do not fit into any of the above categories. c1-c99 is the last variable data column shown in the Data / Matrix Editor. error contains an error code once an error has been made, for use in Try.. Else.. Fentry block. eqn and export are used by the numerical solver (the equation to be resolved to eqn, and this is set equal to exposure if the signed was omitted). ok is set to 1. if a Dialog menu has been exited successfully, and 0. if it was out with the ESC key. 11 and semans2 are the turf for the random number generator used by rank(). sysData is the default data variable used by the BldData command. sysMath stores the result of any graphical calculation (for example, to calculate the derive of a point on the graph) tblStart and the Δtbl used to calculate the table input when it is automatic. tblInput stores the table input when it is not automatic. Page 25 Matrix and Commands A matrix are a rectangular grid of the elements. On the TI-68k calculators, matrix can include any mixture of any staircase (non-list) valid variable type in an expression: You may have matrix of numbers, matrix of strings, matrix of true values or expressions. You can even mix and match type variables – it's perfectly all right there is a string in one element, with a number in the next. There are three ways to enter a matrix on the calculator: Use niche[] brackets: e.g. [a, b, c][d, e,f]] (this is a matrix with row 2 - the range a, b, above the range d, e, f). Use [ ] brackets and semicolons: example [a, b, c;d, e,f] Use { brackets: example. {a, b, c}, {d, e, f}} (this works because the matrix is actually stored as list) You can access a certain element of the matrix by writing the coordinates of the ele you want in [] brackets after reading: matrix[r, c] would have access to inserted in the th ROW and the cth column of the matrix (the matrix is still indexed first by the row, top to bottom, and second by the column, left to right). Also, using a single index - matrix [r] - returns the th row of the matrix as a 1 by # matrix. On earlier calculator patterns, Matrix had the random access property: Access to any element in a matrix took the same amount of time. This was possible because the matrix was enforced in numbers. On the 68k calculators, since can mix types of elements, they have no random access: the calculator has to go through the whole matrix for it to go through an element, so the larger an index is, the longer it takes access. This is not significant for small matrix. But the time keeps increasing linearly, so it can very slowly access the past elements of a large matrix. Except for the constraints of free memory, and at the time it takes access to elements, there is no limit on the number of elements a matrix may have. Tipped as Vector in Mathematics, a vector is a list of our numbers with a geometric representation of n-dimensional space (two representations, actually: as a point in our space—, and as a translation that takes origin at this point). 2- and 3-dimensional vector space are used respectively for 2-dimensional space (the plane), and the usual 3-dimensional space. On the TI-68k CAlculator, matrix and single row, or only one column, are interpreted as vectors for the purposes of the command dotP(), crossP(), and unitV(), as well as the formatted commands ▶Cylind, ▶Polar, ▶Rect, and ▶Sphere. Linear Algebra Common Operations Mathematical commands and extended matrix operators in a linear algebraic way. +, -, and *, for two matrixes, are the corresponding matrix operations (in particular, multiplication matrix is quite complicated). ^ raising a square matrix of an integer power relative by multiplying it alone; if the integer is negative, it takes the inverse first. Matrix has a special operator just to themselves: T, called the transpose operator. It flips the matrix on its main diagonal, so rows become columns and columns become rows. The +, operators -, *, and/can be applied to a square matrix with a staircase as well, by multiplying the staircase by the identity matrix. For multiplication and division, this results in the operation being performed in each element, while addition and subtraction results in adding or subtracting the staircase to each element on the main diagonal. Since periodically you want to perform these element-by-component operations, alternatives .+, .-, . *, ./, and .^ are provided, which do so for two matrix and for a matrix and an expression. The Exponential function and Trig Calculator function provide a special exponential interpretation and trigger applied to matrix, e^() being the most common. These commands require the matrix to be square and diagonalizable, and return an approximative floating-point value. A diagonalizable matrix A is one that can be expressed in the form A =PDP-1, where D and P are square matrix, and D is diagonal - composed entirely of zero except on the main diagonal. If a matrix is diagonalizable, the calculator can computer explicit values for D and P using eigVl() and eigVc():D=Diagnose(eigVl(A)) P = eigVc(A) If a matrix is not diagonalizable, the result of eigVc() will not have an inverse. 19 apply functions such as e^() matrix by first writing the matrix in the PDP-1 form, and then return PF(D) P-1. Here, the function is implemented in D by taking f() in each diagonal element (the elements in the diagonal remain zero). The following definition is used for the following commands: ^, ln(), log(), and root(), $\cos^{-1}$(), Sin(), $\sin^{-1}$(), Time(), and $\tan^{-1}$(). cosh(), $\cosh^{-1}$(), Sin(), $\sinh^{-1}$(), tanh(), and $\tanh^{-1}$() Other Operations on Matrix Most extended mathematical commands in matrix have not been applied to each element; gcd() is a good example. But other commandments behave in unpredicible ways. Commands sort and sort rows and column vectors as if they have been list. Math/Statistical commands list act on matrix as they would on list, which results in a row vector that has the operation done on each column: Finally, there are the commands that are specifically meant for matrix. The following are found in the submenu of matrix in the math popup menu.identity() list▶mat() LU match▶list() mRowAdd() mRowAdd() newMat() Normal QR Conditional Statement, such as If, When(), and Meanwhile, accept matrix of true values as well as single true values. The check will be interpreted as true if and only if each element of the matrix is true, effectively combining each element of the matrix with and. The most common way for matrixes of true values to be created is with relationshipal operators (=, ≠, >, ≥, <, ≤) applied to matrix. They will return the single value 'false' unless both sides of the relationship are matrix of equal size, in this case the matrix will be compared element by element, returning a new matrix. Page 26 Graphics Commands we are dealing with a graphic calculator after all, so it makes sense that there are plenty of graphic commands to talk about. Although the graphical capabilities of the calculator are limited (a monochrome LCD with either 160x100 or 240x128 resolution) these commands allow programmers to squeeze everything from the screen that they can. This page is divided into two sections: Drawing commands for general tasks such as line drawing, circles, or individual pixels, and graphing commands that deal with the graphs of various functions. If you're not interested in the math aspects of graphics, you might want to skip the second section, especially in reading first. However, all functions discussed on this page share a common aspect: the differences from normal graphics. When a graph variable derived, the result is semi-permanent: using a command such as the ClrDraw key graph, or changing some settings, can do nothing more than cause the graph to be reduced. However, all the commands in this generated page trace items disappear permanently when the screen is cleared. If you want to draw an image on the screen, you have two choices. First, you can build it from shapes such as dots, circles and lines. This can work great for a user announcing, but it pretty limits your lot of face logs when you want to draw anything more complicated. So the other option is to have a picture already stored in a variable, and just show it in the screen as you need. We will cover both of these in order. A universally useful cartoon command is ClrDraw: That's clear everything draws on the grafscreen. Point vs Pixel TI-Basic enables two ways specifying a location on the screen: point coordinator and pixel coordinates. The coordinate points are the ones that you see most often: they are coordinated to determine by the grafing window, and (xmin, ymin) being the bottom edge left and (xmax, ymax) the right top. Because the grafing window has to be initialized if you want to use these in a logical way, this is not usually the best type of coordinate to use. The alternative is pixel coordinates (pixel refers to the individual points that make up the calculator screen). The grafscreen is made up of 77 rows and 159 columns of pixels, and 103 rows and 239 columns on a wide calculator (this ignores the right pixels column, which cannot be mapped to). Thus, the pixel coordinates refer to each pixel by the pair (row, column). This is actually reversed in the coordinate points: the first number determines the vertical location, and the second number where horizontally. The rows and columns are counted starting from 0, and the 0th row is the top row. Thus, (0.0) refers to the top left pixels of the grafscreen, and (76.158) or (102.238) refer to the pixels right bottom. For pretty much every drawing order, you have to specify a location on the screen graph in which to draw. You can choose either type of coordinate (although your life will be simple and coordinated pixel) – for each order point, there's a matching pixel order, and vice versa (with an exception we'll get from later). For example, you can draw a line with the line command, and the syntax will be: Or you can use the Pxlline command, with the syntax: PxlLine row1, col1, row2, col2 The following table gives you all the simple drawing commands in the two shapes: Sprites Using Picture Variables, you can store a rectangular image in any size , to be displayed on the screen later. In video game language, such an image is often called a sprit. The several commands that deal with picture variables are covered in depth in the Sprites article (CyclePic described in the Animation article because, you guess it, it allows you to display an animated image). One important thing to remember is that they only use pixels coordinator – another reason to use these. The seven commandments sprite are: Andpic CyclePic NewPic RclPic RplPic RplPic Stopic XorPic The usual route of graphics is to store in a variable equation such as y1(x), and then simply go to the graph screen you can resize the window and other fun stuff). For the programmed convenience, any sort of graph manipulation you can select from a menu is also available as an order. The 68k calculator also provides another way for graphics, probably more appropriate in programming, which draws a graph of a specific expression. These expressions graphene simulate the graph of an equation from any graphical mode. In special note is the Graph order – unlike other commands, which can be deleted with ClrDraw or any other way to refresh the screen, the output of Graph can only be deleted by ClrGraph or by going to the Y= screen to re-enable the normal function =function. ClrGraph Drafunc DrawInv DrawParm zoom in commands to resize the grafing window determined by xmin, xmax, ymin, and ymax. For the most part, these four variables are all changed. The exception is ZoomStd, which restores certain other values to their default, and ZoomSto and ZoomRcl, which back up and restore all window variables. ZoomBox ZoomData ZoomDec ZoomFit ZoomInt ZoomInt ZoomPrev ZoomRcl ZoomSqr ZoomStd ZoomTrig These final commands are more or less miscelane. Unity features are that they deal with the graph of an actual variable (the ones you edit in the Y= screen). Of these, FnOff is probably the one to remember: It closes all derived variables without actually deleting them. BldData FnOff FnOn RclGDB StoGDB Style Table Trace Trace