Correlation

The Custom-Font SDK for Ti-83+ Ti-Basic Programs

Rickie Malgren, aka Hot Dog

Table Of Contents

FOR BEGINNERS:

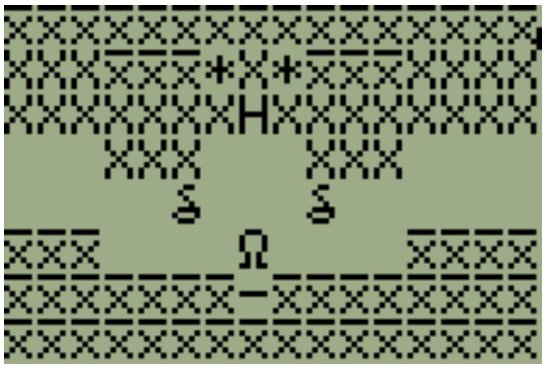
ADVANCED TOPICS:

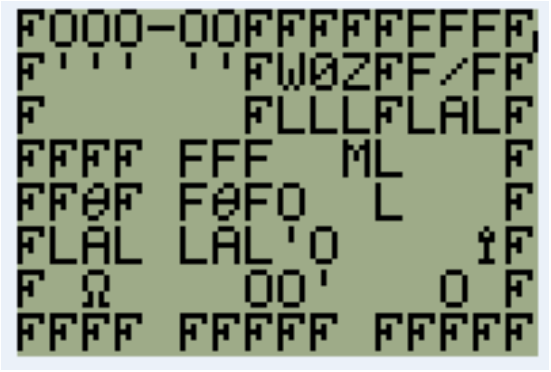
CREDITS:

What is Correlation?

When it comes to the Ti-83+, a lot of games are written in Ti-Basic. Does that mean the games are inferior and undeserving of praise? **ABSOLUTELY NOT!** Some of the greatest games ever written for the Ti-83+ are written in Ti-Basic.

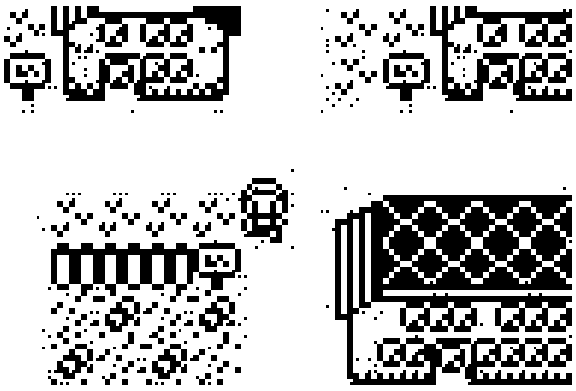
Many of these games are text-based, meaning text is used for graphics. In such games, text can represent rocks, trees, a person, bears, ducks, even water if the programmer wants. Check out this RPG game and notice that there are Os and |s for trees, and Xs for walls.





Sadly, this text stuff doesn't fully resemble trees, rocks, walls, people, etc. Do you know why? It's because a text-based game has to use the Ti-83+ font. And the Ti-83+ font contain mostly letters, numbers, symbols and punctuation. It contains no pictures, just text!

Wouldn't it be nice if you could replace or edit the TI-83+ font? What if you could change all the text in the font into pictures so that when you use Output(or Text(you get something like this?



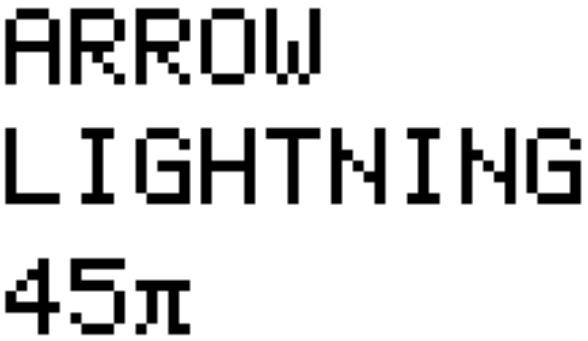
But replacing or editing the Ti-83+ font is hard or impossible, so the next best thing is creating your own font and using it in a Ti-Basic program. And with Correlation, you can do that! You can create a font that uses pictures in place of text. Anything you can do with the Ti-83+ font, you can do with your custom font to create beautiful maps, worlds and game levels. All you do is create a font and enter a few commands, and Correlation handles the rest.

And best of all, your Ti-Basic program, which uses Correlation, will run faster than a similar Ti-Basic program that does not use Correlation. Hard to believe? You'll definitely see a difference as you write processor-intensive Ti-Basic games that use Correlation.

This guide is meant to give you a sure footing in using customized fonts in your Ti-Basic game. So if you're ready, start by looking at the terms on the next page and making sure that you understand what they mean. Or, my friend Rebma Boss prepared a video tutorial on YouTube if you are interested: (LINK COMING SOON!)

Important Terms

Character: A single symbol used in creating sentences. Letters of the alphabet, punctuation marks and numbers are all characters. The Ti-83+ font has a total of 256 characters.



The first line, "ARROW," has 5 characters. The second line has 9 characters. The third line has 3 characters.

It is NOT necessary for your Correlation Font to have 256 characters. You can have as few as 5 characters in your font, or a moderate number of characters such as 110.

ASCII Value: A number that uniquely identifies and defines a particular character. Every character that you can see on your computer screen (or Ti-83+ screen) has a number that distinguishes that character from another character.

<u>Some example ASCII Values:</u>

Uppercase A = 65

0 = 48

& = 38

Lowercase z = 122

Uppercase Letters from A to Z = 65 to 80, one number for each letter

Lowercase Letters from a to z = 97 to 122, one number for each letter

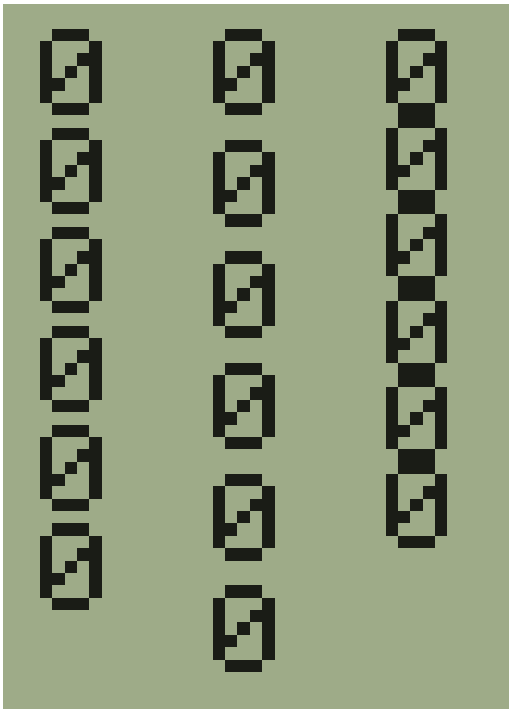
Starting Character Number: The smallest ASCII value allowed by a Correlation font. If you choose a starting character number of 68 in your font, you cannot display characters with ASCII values smaller than 68. If the Ti-83+ font had a starting character number of 68 (the ASCII value for "D"), you would not be able to display the sentence ABC using your TI-Basic program.

Ending Character Number: The largest ASCII value allowed by a Correlation font. If you choose an ending character number of 96 in your font, you cannot display characters with ASCII values larger than 96. If the Ti-83+ font had an ending character number of 96, you would not be able to display any lowercase letters using your Ti-Basic program.

Horizontal Blank Space: The number of white pixels found between two characters. This space keeps characters from running into each other, as well as making text easy to read.

In the first line of 0s, there is one horizontal blank space between each 0. In the second line of 0s, there are **two** horizontal blank spaces between each 0. In the last line, there are NO horizontal blank spaces between each 0, a good idea for fonts that use graphical images in place of text.

Vertical Blank Space: The number of white pixels counted from the bottom of one character to the top of the next. This space keeps lines from running into each other, as well as making text easy to read.



In the first column of 0s, there is one vertical blank space between each 0. In the second column of 0s, there are **two** vertical blank spaces between each 0. In the last column, there are NO vertical blank spaces between each 0, a good idea for fonts that use graphical images in place of text.

Introduction to Using Custom Fonts in a Ti-Basic Program

As you know, when you want to display text in a Ti-Basic program, you can normally use Output(to display homescreen-style text on 16 columns and 8 rows. You can also use Text(to display text on 96 rows and 64 columns.

Whenever you create a custom font and want to use it in your Ti-Basic program, you can still use Output(and Text(to display text using the TI-83+ font. If you want to display text using your own font, you will need to use ln(to display homescreen-style text with a custom font, and e^(to display anywhere-on-screen text with a custom font.

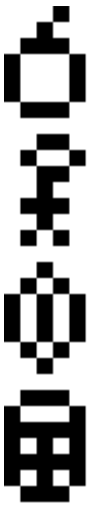
ln(1,1, Str0)

e^(45, 20, sub(Str1,1,2))

Now, I'm really sorry for stating the obvious here, but when you use Output(and Text(, you are telling the calculator exactly what you want it to display on the screen.

Output(1, 1, “ABCFE”)	You will see the string “ABCFE.” You will not see something funky such as 183.!328.
Text(5, 2, “ACORN”)	You will not see the string “DOG,” and you will not see the word “BIRD.” You will see the string “ACORN.”

As you can see, your calculator will display exactly what you tell it to. So let’s say you have this font that you made, with a bomb, a person, a diamond and a can of soda:



It would be nice if you could type in something like

Output(1,1, “ ◆◆◆◆◆◆ ”). But you can’t! That’s because you’re only able to type characters that are part of the TI-83+ font. This diamond symbol in your font is not part of the Ti-83+ font, so you can’t type it using your keyboard.

However, you **are** able to type in Output(1,1, “AAAAAA”). Can you see where I’m going? Normally, the calculator will display what you tell it to display. But with the help of Correlation, you can tell the

calculator to display something DIFFERENT from what you type! In this case, you can tell the calculator to display the diamond from your font every time it reads an A. So ln(1,1, "AAAAAA") would display 6 diamonds. (By the way, remember that you use either ln or e^ to display text using a custom font.)

And you don't need to stop there! Using Correlation, you could tell the Ti-83+ to display the soda pop can (at the bottom of the sample font) whenever your Ti-83+ reads a "B" inside of a string. So ln(5,7, "ABBBA") would display diamond, soda, soda, soda, diamond.

I'm going to call this process **assigning.** Essentially, you assign a character of your custom font to a character of the Ti-83+ font. In the above examples, you assigned a diamond to the letter "A", and you assigned a soda can to the letter "B".

There's a catch when you assign characters from your font to characters of the Ti-83+ font. You can only assign the **first** character of your font, after which the rest of the characters are assigned automatically, in the order the characters are drawn.

Here's the deal. Taking a look at the example font on page 8, let's say you assign the bomb to a particular character of the Ti-83+ font—we'll refer to this Ti-83+ character as Character 1. With the bomb assigned to Character1, the person in your font will be assigned to the character after Character 1. The diamond in your font will be assigned to the character **two characters** away from Character 1. As you might expect, the soda-can will be assigned three characters away from character 1.

So let's say you assign the bomb to the letter "T." Then every time the calculator reads the letter T in a display routine, it will show a bomb. If the calculator reads a "U" inside of a string, it will show a person.

The calculator will show a diamond if it reads a V, and a soda can when it reads a W. ln(1,5, “TTWUV”) translates to displaying bomb, bomb, soda, person, diamond.

So, you’re probably asking by now, “If I know what TI-83+ character to use to display the first character of my font, how will I know what characters of the Ti-83+ font to use to display OTHER characters of my personal font?” The answer is simple: use the table I have given you, called “Ti-83+ Character Codes”. This page provides a bunch of characters you can type, from left to right, top to bottom, in the order they appear in the Ti-83+ font. Since these characters are in order, you can look at what you assigned the first character of your custom font to, and find out other characters from there.

Suppose that you look at the table and assign the “bomb” in your font to the number 8. Then Correlation will automatically assign the person to the number 9. Whenever you use Output(with a negative sign in the string, you will see a diamond instead of a negative sign. And finally, E will be replaced with the soda can. e^(0,0, “8989-E”) will display bomb, person, bomb, person, diamond, soda can.

By the way, be aware that there are some ASCII values on the Ti-83+ that you cannot type characters for. These are marked with black boxes on the table.

Designing a Font

Creating your own font is relatively straight forward, but there are some considerations you have to take into account before you begin the drawing process. You will need a computer with a program that can create monochrome bitmaps. For computers equipped with Microsoft Windows, I highly recommend Microsoft Paint.

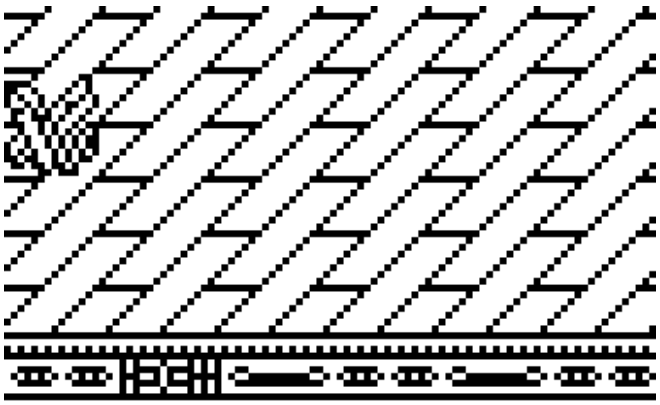
The first thing you want to consider is how big you want each character in your font. The smaller the characters in your font are, the more characters you can display on screen at once. However, small characters aren't as detailed, and require bigger strings in your Ti-Basic program. Larger characters are more detailed and require less string space in your Ti-Basic program, but you can't display as many distinct characters on screen at once.

The standard Ti-83+ Homescreen font has characters 5 pixels wide and 7 pixels high. (From this point onward, I will call this font a 6x8 font because there's one horizontal blank space and one vertical blank space for each character. This technically means each character needs its own space 6 pixels wide and 8 pixels high) In a 6x8 font, characters are somewhat detailed, but you can't display a lot of text on the home screen. On the other hand, the Ti-83+ "Small Text" font has characters that average to 3 pixels wide and 5 pixels high. This means the characters are low on detail, but one can see more text at once.

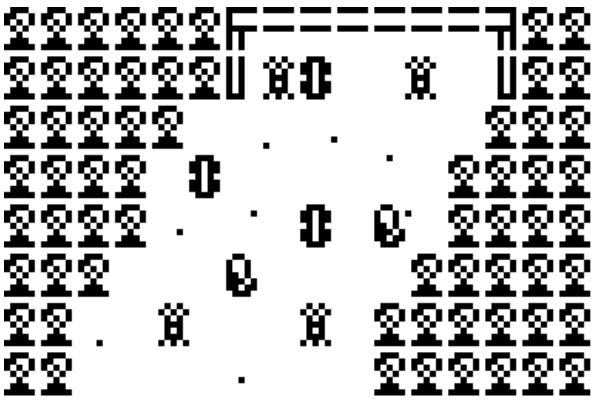
So if you want a game with detailed images, you should use a large font—16 wide by 16 high is very common for this. If you want a game that will display more, you'll want a smaller font. A lot of people use 8

wide by 8 high characters for this purpose. **I have found the 6x8 homescreen font to be a very, very common font size as well.**

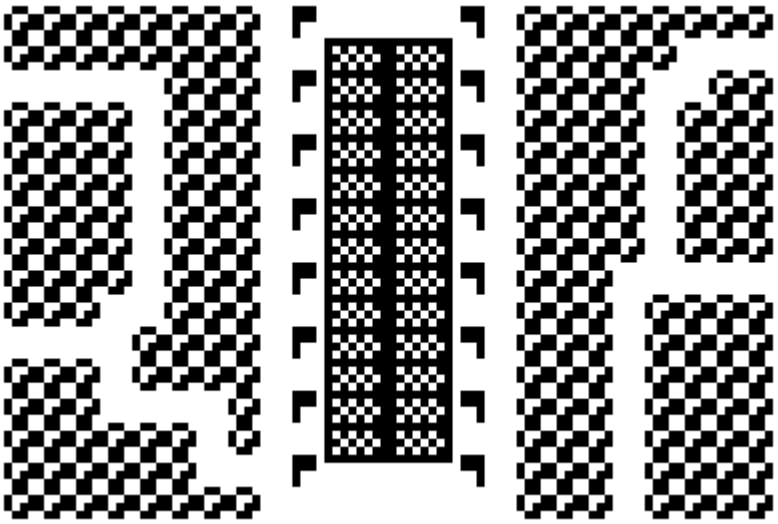
Below is a game using 16 x 16 sized characters as a font size with no blank spaces between characters. Notice that only 24 characters—6 across and 4 down—can be displayed at once.



The next screenshot is another game using the common 6x8 sized homescreen font size, with a horizontal blank space and a vertical blank space between each character. Notice that it can display 12 characters across and 8 characters down, although the detail is slightly less because of smaller characters. By the way, notice what horizontal and vertical blank spaces do to the checkpoint at the top of the picture—it looks like someone took a chainsaw to it and cut it into several pieces.



Now for a 4 x 4 font with no blank spaces between characters. Although the detail is almost non-existent and the string size is large, one can display 24 characters across and 16 characters down. Notice how finely detailed the white pathways are because of how small the tiles are.



All characters in your font must be the same size. Furthermore, your characters can be no bigger than 16 pixels wide and 16 pixels high, nor can it be smaller than 2 pixels wide by 2 pixels high.

Once you decide on the font size that you would like to use, you will need to decide if you want any horizontal or vertical spacing between the characters. As a good rule of thumb, you will want to have horizontal and vertical spacing for any text-based characters you place inside of your font, so that your text is easy to read. If there are graphical-based characters inside of your fonts, you will usually want to avoid having any spacing so that your game looks like a complete image. This is not a hard-and-fast rule, however, just a suggestion. Notice on the above image, for example, there's an intentional blank space between the flags and the road because it makes the world look nicer and clearer. Similarly, on the image of the racecar track, blank

spaces help us in determining the difference between the road block sign and the car next to it.

In this sample font, all letters of the alphabet are separated by horizontal and vertical white spaces (represented as red lines in this picture). That way text can easily be readable. (DO NOT add red to your custom font, by the way, no matter how many times you see it in the examples.)

With the graphics, however, there are NO blank spaces between characters of the font. When you create a world with dirt, rivers, signs and roads, you want everything to look seamless. Remember that at the bottom of page thirteen, the "checkpoint sign" on the racecar track had horizontal and vertical blank spaces, and it did not look pretty whatsoever.

So keep these ideas in mind. Once you decide on a font size (and for practice, I highly recommend 6x8 or 8x8), open up your paint program. Start by setting the width of your bitmap to the width of your choice. Your height should be the height of each character, multiplied by the number of characters you want. Notice from the sample font that characters are placed one after the other in a straight line.

When drawing the characters inside of your font, each character requires its own "area" for lack of a better word. In a 6 x 8 font, every character requires space 6 pixels wide by 8 pixels high. You'll want to start at X = 0 and Y = 0 on the bitmap to draw your first character. Your next character will be at Y = Character Height, X = 0. Then the character afterwards will be drawn at Y = Character Height * 2, then

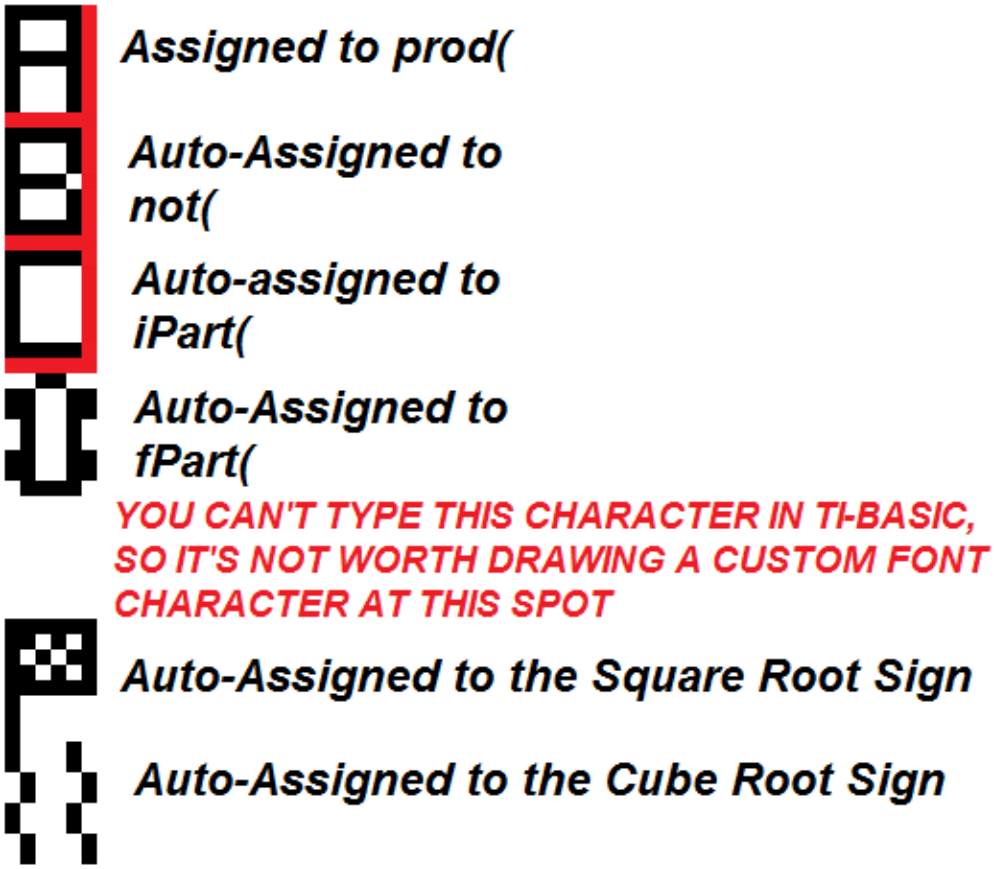
Character Height * 3, and so on. Look at the sample 6x8 font at the top of the previous page. The character “A” is drawn at Y = 0. The character “B” is drawn at Y = 8. The character “C” is drawn at Y = 16. The car is drawn at Y = 24.

As you draw your font, remember to include blank spaces where appropriate. If your next character requires horizontal blank spaces, that character should be thinner than the width of your bitmap. Suppose you are creating a font 6 pixels wide. A character with a desired horizontal blank space of 1 should be 5 pixels wide with one pixel of white space. A character with a desired horizontal blank space of 2 should 4 pixels wide. Of course, a character 6 pixels wide on this particular bitmap will have NO horizontal blank space.

When taking into account vertical blank spaces, the same rule applies similarly. Let’s say that your font consists of characters no bigger than 8 pixels high. A character of text should have at least one vertical blank space, so you would want to make the character only 7 pixels high. Then you will need to make sure there is one pixel of blank space between the bottom of the aforementioned character and the top of the next character. For graphical characters on a font with 8-pixel-high characters, you’ll want to consider NO WHITE SPACES between most—if not all—of the characters. Yes, that means placing your graphical characters right on top of each other.

IMPORTANT: Be aware of what Ti-83+ character you wish to assign the first character of your custom font to. Suppose you assign the letter “A” in the previous page’s example font to the Ti-Basic word prod(. Looking at the table of Ti-83+ characters, ln(3,4, “prod(not(iPart(fPart(”) will display the A, the B, the C, and the car. But what about the checkered flag? You can’t enter a character that will display the checkered flag, so you will never be able to display the

checkered flag! In this case, you'll want to force Correlation to assign the checkered flag to the next available character, the square-root sign. You can do this by making blank the character that Correlation would have assigned to the Ti-Basic character you can't type.



After you have finished drawing all the characters that you would want to display in your Ti-Basic game, there might be some unnecessary white space at the bottom, space from making the height of your bitmap too big. You'll want to clip this off, but remember to leave vertical blank space for you bottom character if it requires vertical blank space.

Make sure your font has all the characters and blank spaces that you want. After words, you'll want to adjust the width of your bitmap as follows: If your font is 8 pixels wide or less, the width of your bitmap

should be adjusted to be **exactly** 8 pixels. Otherwise, your bitmap width must be 16 pixels. Save your bitmap as a monochrome bitmap and DO NOT COMPRESS IT. Also, since this will go to your calculator eventually, **the first character of your filename should be a letter, and the rest should be 7 letters or numbers**. Your filename must also be *at least* 8 characters long. A8SEIFILCMWOW.bmp is a valid filename, 3_SL3__.bmp is not.

That's it! Your font is done! However, you can't put a bitmap onto your calculator, so read the next chapter to learn how to compile your font into a Ti-83+ program that CAN be placed on your calculator.

Compiling a Font

Correlation comes with a CLI font compiler and a GUI font compiler, both of which let you compile a monochrome bitmap font into a Ti-83+ program that you can use with Correlation. Both programs are available in case one works and the other doesn't. You will need java, and you will also need to make sure that your bitmap and the programs are in the same folder.

Running CorrelationFontCompiler.jar is straight forward once you know how to run CorrelationFontCompiler.class, so I will mostly provide instructions for using the CLI version of the compiler. Open a CLI, such as Windows Command Prompt, and go to the directory where the program is located. Type in java CorrelationFontCompiler.

Under "bitmap image to parse," type in the name of your bitmap, including ".bmp". Note that your bitmap should be in the same folder as CorrelationFontCompiler.class.

For "width of each character," type in how many pixels wide each character of your font is. For "height of each character," type in how many pixels high each character of your font is.

For the next step, you will see "Starting Character Number." On the table where you see all the TI-83+ font characters, choose the character that you want to assign the first character of your font to. Then use the columns and rows to find its hexadecimal value. (For example, "H" is 48, since it's on column 8 and row 4) Convert this number to decimal and enter this as your starting character number.

Now take this number, subtract 1, and add the number of characters in your font. This will be your “Ending Character Number.” Enter this in.

That’s it! Your font will compile to a Ti-83+ program, and the name of the program will be taken from the name of your bitmap.

If you are using the GUI version of the font compiler, you must specify a name for your program. Remember that the name must be 8 characters long, and a valid program name. Sorry, you cannot use Theta in the name.

Using Your Font in a Ti-Basic Program

The important thing to remember about Correlation is you can display custom-font text in a manner very similar to displaying Ti-83+ text in a Ti-Basic program. There are some lines of code you have to add to your TI-Basic program, but other than that the transition to Correlation is very smooth, mostly turning one instruction into another. ***So take note, if there's a great Text-Based Ti-Basic game that you want to convert to use a custom font with Correlation, it is not hard to do so, just read the next chapter for more information.***

If you want to use Correlation, you require two programs: pgrmCORELATE and the Corelate application. pgrmCORELATE must be stored in RAM. Your font(s) can be stored in either RAM or in the archive.

You must have the line 1:Asm(pgrmCORELATE before Correlation will run in your program. You don't have to start your Ti-Basic program with this line, but it's not a bad idea to do so. This line prepares the Ti-Basic program to use your custom font.

Also, you must have the line 0:Asm(pgrmCORELATE before the end of your Ti-Basic program. Thus, if there are several areas where your program could end, you must have 0:Asm(pgrmCORELATE at each of those spots. This line of code closes Correlation so that your calculator can run normally again.

While you can use several fonts in your Ti-Basic program, you can display text using only one font at a time. Use the instruction **int(** to select your font. int(must be followed by the eight characters of your font name, with ONE quotation mark at the beginning. (DO NOT put quotation marks at the end.) For example, if your font is stored in pgrmTESTFONT, use int("TESTFONT to select that font for displaying text. You can use int(at anytime during your program to select another font.

Remember that you use ln(to display text in a similar manner that Output(does, and you use e^(to display text in a manner similar to Text(. ln(will only work with a 6x8 font, but Text(can work with any font size up to 16 x 16.

The last thing that you need to know for right now is that after you use ln(and e^(, you will not see your text immediately. Normally, when you use Output(and Text(, every line of text drawn is shown immediately, and that slows your Ti-Basic program down. Correlation takes care of this by not displaying text drawn with ln(and e^(until you ask it to. Rather, it prepares it, so that when you are done drawing all your text and use int("DD to display it, all text will show up at once. This will greatly increase the speed of your program.

You can also display text ready to drawn by using any of the TI-Basic drawing and graphing commands, such as Pt-On(. Note that if there's any text you would not normally redraw in a Ti-Basic program, you do not need to redraw it with Correlation.

Converting an Old Ti-Basic Game to Use a Custom Font

If someone has written a game that uses the homescreen, you can create a 6x8 font to replace the one for their game. While you can do the same for a game that uses text on the graph screen, it's a little harder. This is because all characters in you font must be the same size, whereas the Ti-83+ graphscreen / small font has characters of different sizes. Caution is recommended when converting a Ti-Basic game that uses the small font. I also advise using a 3x5 font for this purpose.

To convert an older Ti-Basic game, remember to include 1:Asm(pgrmCORELATE) wherever the program starts, and 0:Asm(pgrmCORELATE) wherever the program ends. Do not forget, in addition, to use int(to specify the font that will replace the default TI-83+ font.

Now comes the fun part: Search the program for any lines with Output(or Text(. If you find Output(and you want it to display its contents using the default homescreen text, *leave it alone.* Same with Text(. However, change Output(to ln(if you want to display homescreen text using a customized font. Also, change Text(to e^(for any text that you want displayed using a customized font.

Finally, remember to use int(“DD when you are ready to display any text that was drawn. As was aforementioned, any text that you draw will not be seen until you use the command int(“DD.

Tips, Tricks and Optimizations

- The **Run Indicator** is the little bar that you see at the upper-right hand corner of your screen sometimes, the bar that says your calculator is busy. (Screenshot)

  You can turn off the Run Indicator by placing int("RN in your Ti-Basic program. Use int("RY to turn it back on. Note that you should turn it back on when your program exits.

- While sub(is useful for getting substrings, you can use abs(to get substrings for strings that you use with Correlation. Abs(works much faster than sub(does. Abs(Str0, 4, 20) will take 20 characters from Str0, starting at the 4th character. Make sure that your string does not have any illegal characters (meaning characters not found on the TI-83+ character table), or you will get funky results. Just like with int, you can use abs(for math by specifying only one parameter.
- If you want speed in your program, fonts 16 pixels wide will display the fastest, hands down. However, fonts 8 pixels wide (or 6 pixels wide with ln) have decent speeds as well.
- It's a well-proven fact that when you design a font for a calculator, your Ws and Ms must be at least 5 pixels wide, or people won't be able to tell that they are Ws and Ms. For a font with letters less than 5 pixels wide, you can tell Correlation to make Ws and Ms wider than the rest of the characters. For example, take a look at the example font below:

This font has characters 3 pixels wide, but you can tell Correlation that the W and M MUST be 5 pixels wide. (This is called **WMMode**.) Notice in the picture that the W is indeed 5 pixels wide. To turn on WMMode, use the command int("WY. Use int("WN to turn off WMMode. (screenshot without WMMode, and with WMMode) Be careful, your W must be assigned to the Ti-83+ W and your M to the Ti-83+ M for this to work.

- There are only so many characters available on the Ti-83+ keypad. With the fact that you might be forced to click menu after menu to retrieve values such as product(and solve(for some font characters, a font with over 200 characters can be hard to work with. A huge font is indeed necessary for some games, but whenever you can, try to have several fonts using characters A-Z (and other easy-to-reach characters) as opposed to one big font.
- int(is the only Ti-Basic command that has a strict format in Correlation. Other than that, if there are optimizations you like to use in Ti-Basic programs, you can use those same optimizations with Correlation commands.

Error Messages

Occasionally, you may come across an error message that you've never seen in Ti-Basic before. These error messages are provided by Correlation to help you find errors that you may have made. This section of the guide will explain error messages and the mistakes that you should fix inside of your program as a result.

ERR: MISSING APP

You will receive this error if your Ti-83+ is missing the application Corelate. To use Correlation, Corelate MUST be on your calculator.

ERR:NO SUCH FONT

You are attempting to use int(to load a font that doesn't exist. Make sure that you typed the font name correctly and that the font is indeed on your calculator.

ERR: OPTION

Before Asm(pgrmCORELATE, you must have either 1: or 0:, such as 1:Asm(pgrmCORELATE. Also, DO NOT end the line in parenthesis.

ERR: DOMAIN

Okay, so you've seen this error before. Just make sure that you specify acceptable values inside of your function. I'm reminding you of this error because there are ranges for parameters you may have never encountered before.

ERR:INT(

int(is the one command that requires a strict format if you use Correlation. If you specify an invalid parameter or if you put quotation marks at the end, you will receive this error.

ERR:OUT OF RANGE

Every font has a range of characters you can display using strings. This range is from your Starting Character Number to your Ending Character Number. If you attempt to go outside of this range, you will receive this error. Correlation will return an error, for example, if you try to use an "!" for a font that only allows A-Z.

ERR: WHICH FONT?

You will see this error if you are attempting to use ln(or e^(to display text without having selected a font first.

ERR: WRONG SIZE

When you use Ln(to display strings, the characters inside of your font must be 6 pixels wide by 8 pixels high. You must select another font to use if the characters do not follow this format, or use e^(instead.

ERR: NOT ALLOWED

You are using a character inside of your string that you are not allowed to use to display a custom font. (Refer to the Ti-83+ Character Table) For example, Select(is not something you can use to display a character from your custom font. This error usually comes from mistyping a character.

ERR:INVALID FONT

You are using int(to select a program that isn't a compiled font.

Special Ln(and e^(Display Modes

*These are advanced techniques to use when you understand the basic concepts of Correlation. I do not recommend these techniques for people using Correlation for the first time. Also, I also recommend they only be used when you are making a fresh TI-Basic game—**using them on an already-made TI-Basic game is strongly discouraged.***

Text can be displayed in one of four different ways in Correlation: Clip Mode, Wrap Mode, Word Wrap Mode and Map Mode. If you've done a lot of Ti-Basic programming, you are most likely familiar with clip and wrap mode, even though the names are somewhat unheard of.

In **Clip Mode**, text will stop displaying once the edge of the screen is reached. If you ever used Text(in a Ti-Basic program, the displayed text uses clip mode. Use int("M0 to start displaying text using Clip Mode.

If text is displayed using **Wrap Mode**, characters will be drawn until a line is full, after which the characters will continue onto the next line on the screen. If you ever used Output(in a Ti-Basic program, it normally uses Wrap mode. This is also the default mode used for ln(and e^(. To display text using Wrap Mode, include the line int("M1

Word Wrap Mode is a special mode that takes a string and draws it Word Wrap style—that is, moving text from line to line to create an

easy-to-read paragraph. The game will then pause and allow the user to scroll text up and down until enter or ON is pressed. This allows a user to create storylines with long text without having to make a bunch of complex calculations. Use int("M2 to display text using Word Wrap. Word Wrap Mode cannot be used on fonts with characters larger than 8 pixels wide; attempting to do so will return an ERR:WRONG SIZE.

For Word Wrap mode, you must have ASCII character 41 in your font to allow spaces, and you also must have ASCII character 91 in your font (Theta on the Ti-83+ character table). That means the starting character value in your font must be 41 or less, and your ending character value must be 91 or more.

When displaying a string using Word Wrap, use the Ti-Basic command "and" in your string to represent a pause, that is, when you want the user to scroll text and/or press Enter and On. You can use "and" between sentences to separate text so that the user has to press Enter to view the next section of text. When your string contains all the text you want to display, you must end it with "and" and the letter theta.

Word Wrap lets a user press the down key up to 50 times for a single section of text. If the particular section of text you are displaying requires the user to press down more than 50 times—that is, if your text takes more than 50 lines—you will receive an error message when the user presses the down key for the 51st time.

Map Mode is another special mode that is meant for games such as Text-Based RPGs. Normally, you probably used multiple strings and/or sub(to display a text-based map that you could move around. However, with Map Mode, you only need one string (containing the text you would use to display a map) and a map-width, such as "30 characters wide." Take your map data and place it in your string, one

row of data after another so that all the rows of your maps are placed together in one string. Correlation handles the rest so that your map comes out right without any weird text wrapping or clipping.

If you have these strings for a map with 7 rows:

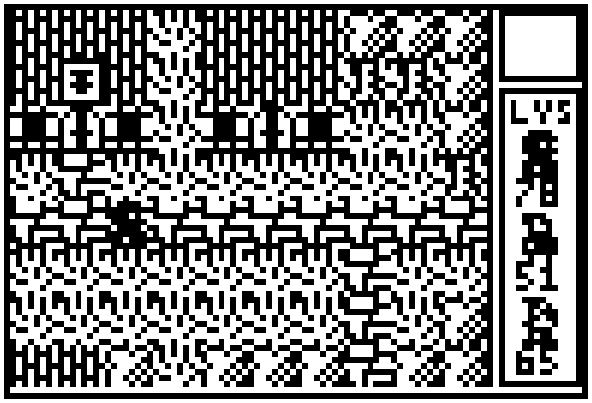
```
ABBCADED
EIDLSKWS
GIELDDDD
ISOE
DKE
AO      D
A    SDFD
```

Put them together in one string like this:

```
“ABBCADEDEIDLSKWSGIELDDDDISOE    DKE     AO      DA    SDFD”
```

Use int(“M3 to enter Map Mode. Store a value into ThetaStep to specify a map width, which cannot be bigger than 99. Note that the height of your map does not need to be specified.

For all methods of displaying text, you can specify a window for text to be displayed in. Any text (or portion of text) that is drawn outside the window will not be displayed. Use ThetaMin for the left-hand coordinate of your window and ThetaMax for the right-hand coordinate of your window. Use TMin for the top of your window and TMax for the bottom of your window.



Here, the map doesn't draw over the entire screen in this screenshot. That way, the status bar on the far right won't be erased. For this window, ThetaMin = 0, ThetaMax = 80, TMin = 0 and TMax = 63.

If you change even one window coordinate or the width of a map (with ThetaStep), even for the first time, you must use int("FF to tell Correlation to make the proper adjustments.

Advanced Text Graphics

*These are advanced techniques to use when you understand the basic concepts of Correlation. I do not recommend these techniques for people using Correlation for the first time. Also, I also recommend they only be used when you are making a fresh TI-Basic game—**using them on an already-made TI-Basic game is strongly discouraged.***

By default, any text that you draw will erase anything underneath it. You have probably seen this when you used Output(and Text(in your old Ti-Basic programs. This method of displaying text is called OVERWRITE, because the text erases/overwrites everything underneath it.

When you draw text from a font with graphical characters, however, you might want to have some transparency so that you don't erase the background. Correlation allows you to draw text with transparency whenever you use ln(and e^(.

Suppose you want black pixels in your font to be drawn and white pixels to be transparent. (This is called OR.) Just include the line int("E1 , and all text afterwards will be drawn with this form of transparency.

```
int("E1
ln(2,7, "EISFE")
```

e^(45,12, Str0)

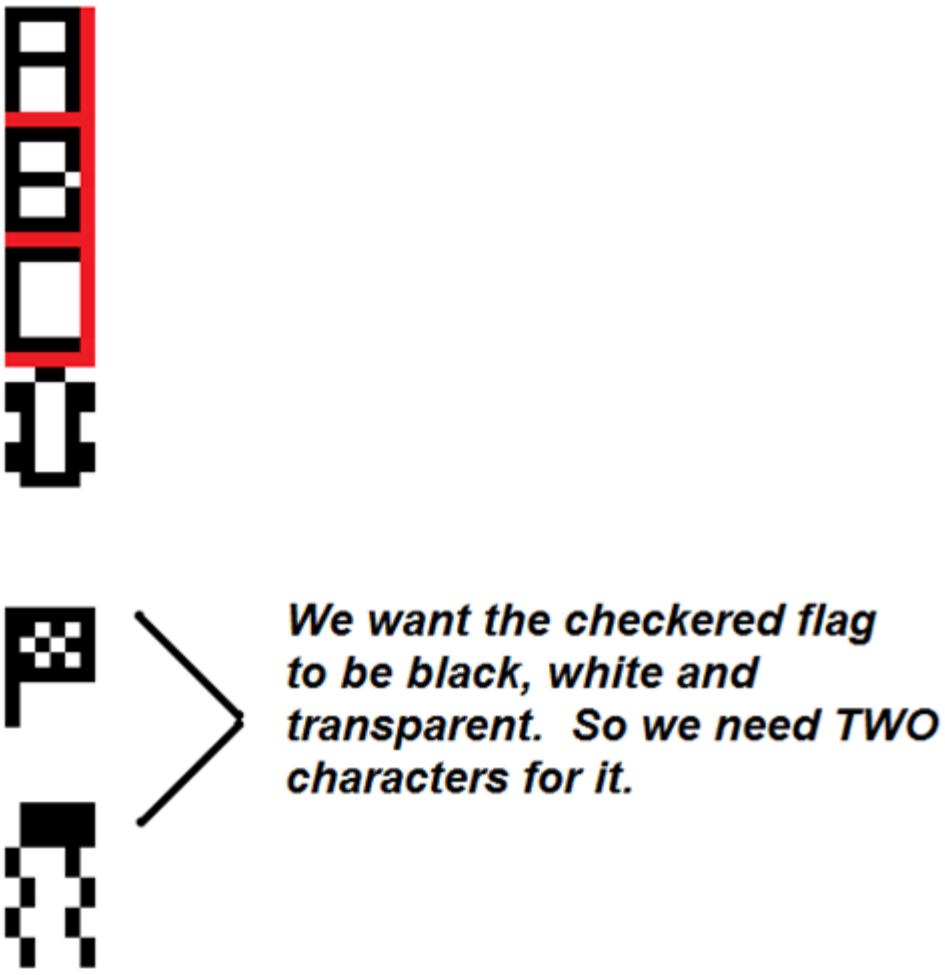
If you include the line int(“E2 , your font will be drawn with an XOR method. White pixels will be transparent, and black pixels will INVERT the colors of anything underneath them.

If you include the line int(“E3 , your font will be drawn with an AND method. White pixels will be drawn normally, and it’s the black pixels that will be transparent.

To switch back to OVERWRITE mode, use int(“E0 . OVERWRITE mode is the default method for drawing text with Correlation.

Sometimes you’ll have a graphical character in your font where you want some parts white, some parts black, some parts inverted, and some parts transparent. If you want to do that, you need to use int(“E4 .

This method is called drawing a MASKED font character, and the method is usually used for drawing a graphical image where you want black, white and transparency all at once. However, if there’s a graphical character you want to draw MASKED, you have to make some adjustments to your font. A Masked font character requires a regular character and a mask to go with it. So two character’s worth of bitmap font space is required—the first space for your regular character, the other space for its mask.



For the first part of your masked font character, any part that you want white or transparent should be colored white, and any part that you want black should be colored black. For the second part of your character, paint any transparent part black and any other part white. Notice in the example above that in the “mask” part of the checkered flag in the font, the transparent area is black and everything else is white.

Masked text draws slower than other text, so don’t abuse masked text. Believe me: you don’t need a lot of masked graphics for a game.

Font Collections

These are advanced techniques to use when you understand the basic concepts of Correlation. I do not recommend these techniques for people using Correlation for the first time.

Sooner or later you may plan on making a game that requires ten or more fonts. Using int(to specify the 8-character name of each font takes a lot of program space, especially when you need to switch back and forth between fonts.

An alternative method is to create what's called a "**font collection**." Suppose you were to create a Zelda game that required 15 fonts. What if you gave the 15 fonts these names?

ZELDAF00, ZELDAF01, ZELDAF02, ZELDAF03, ZELDAF04 … ZELDAF13, ZELDAF14

Notice that these fonts all end in two numbers. So there's a catchy way to select a font using much less program space. Start by using int(to specify the full name of the font that ends in two 0s.

int("ZELDAF00

After that, you can select any of these 15 fonts using only the two digits at the end.

int("01

int("13

Note that if you start your program with, for example, int(“03, Correlation will not know that you are using a font collection! You must always start with the full name of the first font in your collection, the font with two 0s at the end. You will receive an ERR:WHICH FONT? if you use int(with a two-digit number without specifying the first font in your collection.

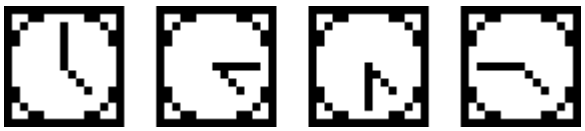
You will also receive an ERR:NO SUCH FONT if you use int(with a two-digit number that does not exist in your font collection. In our example of Zelda, int(“15 would return an error.

A font collection can contain up to 100 fonts, ending in 00-99. These fonts do not need to be the same size or height/width as each other.

Animated Text

These are advanced techniques to use when you understand the basic concepts of Correlation. I do not recommend these techniques for people using Correlation for the first time.

You can design your font so that some of the characters are animated. Animation is often used for effects such as water and blinking lights.

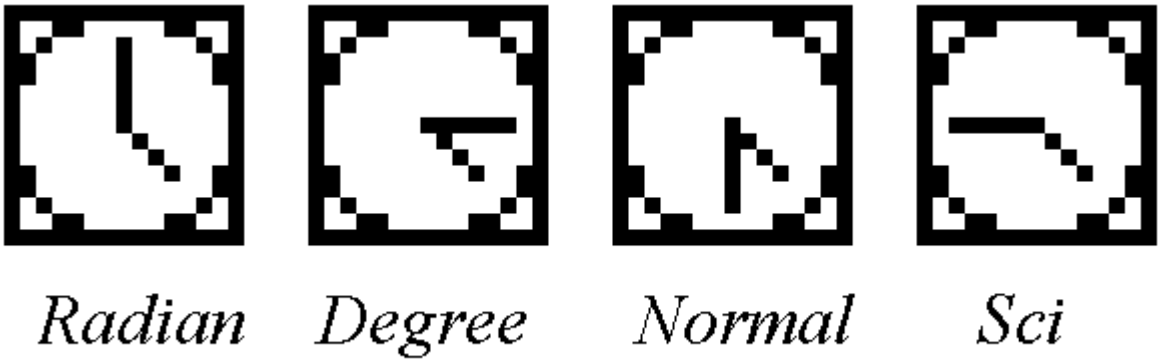


In the above screenshot, the clock is animated so that the big hand moves. Notice that there are four different images of the same clock, four **frames** used to animate it and make the big hand move clockwise. If we were to take these four images and arrange them into a bitmap for a Correlation font, it might look like this:

In fact, that's just how you create animated text for Correlation. You take a series of images that you want to run in succession, rotated over and over again. Then you put them on top of each other in the

bitmap, in the order you want them to run. Punch in a few lines in your Ti-Basic program, and bam! Correlation does all the animation for you.

Animation only works with characters assigned to Ti-83+ characters from Radian to Pxl-Change(. Once your images are aligned from top to bottom in the order that you want them animated, you use the character that the FIRST frame of your animation is assigned to. Suppose, for example, that the images of the clock animation are assigned to these four characters:



Then you would use e^(0,0, “Radian”) to display the clock at the upper-left hand corner of the screen. Your next animation would be assigned to Eng, since Eng comes four characters after Radian (Eng proceeds immediately after Sci). While Correlation handles all animation for you, it is your responsibility to know which character to use to display a particular animated character.

If you were to use int(“DD at this point, your clock would be standing still, and the big hand wouldn’t move an inch. Don’t worry, Correlation knows that the clock should be animated. Every time you use int(“DD, your animation will advance one frame. So by displaying your map over and over in conjunction with int(“DD, you will see an animated clock.

```
Lbl AN
e^(0,0, “Radian”)
int(“DD
Goto AN          ;Every time you Goto AN, you will see a different
                  image of the clock, as if the clock is moving.
```

You must use int(“AY to turn animated tiles on, and int(“AN to turn them off. In addition, you will need to tell Correlation how many frames you want your animation to be. Our clock animation took four frames, but sometimes you may want 2 frames, 16 frames, or even 1 frame (meaning your image doesn’t animate, just in case you want to use ZoomSto, Radian or anything similar to draw a non-animated character of text). To let Correlation know how many frames each animation should take, use det(. Your input will be a list of numbers telling Correlation how many frames each animation takes. Suppose that you have 4 animations: A clock with 4 frames, a running dog with 3 frames, a tree that does absolutely nothing (1 frame), and a traffic light with 4 frames.

```
det({4,3,1,4})          A total of 12 frames, by the way
```

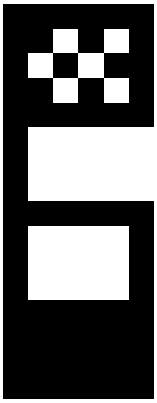
You have a limit of 64 total frames to use for your Ti-Basic program. There is, however, one exception. The very last animation you create in your bitmap can have up to 99 frames. So if you use up the characters assigned from Radian to Pxl-Off(, you can use Pxl-Change to draw a 20 character animation if you want to do so.

Advanced Tips, Tricks and Optimizations

*These are advanced techniques to use when you understand the basic concepts of Correlation. I do not recommend these techniques for people using Correlation for the first time. Also, I also recommend they only be used when you are making a fresh TI-Basic game—**using them on an already-made TI-Basic game is strongly discouraged.***

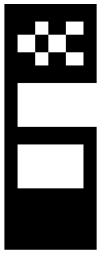
- You can specify a negative value for X and Y coordinates with ln(and e^(. Your negative values can be as low as -9999. This method will also display text faster than when you had to use sub(for strings displayed with the normal Ti-83+ text. Note that a value of 0 in ln—and ONLY ln—will translate to -1, and a value of -1 will translate to -2, and so on.
- If you have a library or asm program such as Xlib, AppVar or Celtic III that you want to use in your Ti-Basic program, you can use it in conjunction with Correlation. However, you must temporarily switch from Correlation to your other library. Use int("CN to turn Correlation off, allowing you to use another library. Use ZoomSto to turn Correlation on again. Be aware that Correlation does not allow you to use, for example, Xlib and Celtic III with each other.

- If you are creating a mask-able character, you may occasionally come across a situation where you want parts of your masked character to **invert** pixels underneath when the character is drawn. To do this, paint black this area in BOTH your normal character and its mask.



The outline of the flag, as well as the flag pole, will now invert any pixels that are underneath them when the checkered flag is drawn.

- Remember that masked characters require space in a font equal to 2 characters. The mask can be placed on a spot that you cannot normally type a character for. Look at the Ti-83+ Character table and observe that you cannot type in the character that comes immediately after the colon, located at 3E on the table. However, when you are designing your font, if you place the mask of your masked character in that spot, you can use, for example, ln(0,0, "::"), and your masked character will display without any issues or error messages.



Assigned to :

There is nothing you can put in your string to display this character, but it works fine as a mask

Assigned to and

The same concept applies to animated text. The first frame of each of your animations MUST be placed on a spot that you can type a character for, but any following frames can be placed in spots that you cannot type characters for.

- int("DD is good when you want to display text you drew all at once. However, int("DD updates the entire screen. If you are writing a game where only one character is drawn over and over without erasing the background, it is faster to update that one character rather than the entire screen. You can use int("EY to tell Correlation to display a character the moment it is drawn. Use int("EN to turn this off, meaning Correlation will not display text until it encounters the line int("DD.
- Occasionally you may have written a Ti-Basic game where you draw a single character of text and have to erase it constantly so that it looks like it is moving. For instance, you might have used Output(1,1, "A") to display the letter A, pretending that the letter A was a person or something. Then you would have used Output(1,1, " ") to erase the A before drawing the A somewhere else. However, if you want, Correlation can do all the erasing for you. You can draw your A, then Correlation will erase it when you want to move the A somewhere else. Just use int("BY to ask it to do so, as well as int("BN to stop. Note that int("EY and int("BY can be used in conjunction with each other.

Advanced Error Messages

ERR: TOO LONG

Word-wrapping only works with text that takes no more than 50 presses of the down key to view. You will need to shorten your text—or change your display window (to show more text at once)—if it is larger than that. Lines are counted as you scroll the text down, so make sure you test for an error by scrolling to the bottom of the string that you displayed. You only need to check for this error once per string that uses Word Wrapping.

ERR: NO SPACE

When you use Word Wrap mode, your selected font must have a starting character value 32 or less. Spaces are used to help Correlation in arranging your text into a readable paragraph.

ERR: NO THETA

When you use Word Wrap mode, your selected font must have an ending character value 91 or more. Theta tells the calculator to stop displaying new sections of text.

ERR: BAD WINDOW

ThetaMin and TMin must be strictly less than ThetaMax and TMax. You also cannot specify any negative coordinates for these parameters. These four coordinates cannot extend beyond the boundaries of your screen. Finally, ThetaMin and ThetaMax can be no less than 24 apart, and TMin/TMax can be no less than 24 apart.

Introduction to Correlific Mode

As you can see, Correlation provides a wide variety of options for making a fun and detailed game. But what's that I hear? You want faster games? Ok, you can always use Correlific Mode. Correlation has a built-in game engine that you can supply commands and data to in order to make a fast game. However, Correlific Mode is a huge topic in and of its own, so I have provided a separate manual. If you want to use Correlific Mode to create even better games, take a good look at the manual, and be aware that you will need some spare RAM on your calculator for the game engine. And I advice you not to use Correlific Mode unless you want a really fast game, because although it is easy, it takes a lot of time and planning to put such a game together.

Special Thanks and List of Testers

SPECIAL THANKS TO THESE PEOPLE FOR MANY DIFFERENT THINGS, BUT ESPECIALLY THE FOLLOWING (IN NO PARTICULAR ORDER):

thepenguin77—General parser help, converting BCD to decimal

calc84maniac—Converting BCD to decimal

Sean McLaughlin—Sprite Routines

Iambian, calcdude84se—Guidance on drawing overwritten sprites

Xeda112358, BrandonW, KermM—General parser help

DJ Omnimaga—Countless suggestions and hints that helped make this a high quality package

Graphmastur—Special hook help, idea for Font Collections

BuckeyeDude—VAT and variable help, general parser help

Deep Thought—Main Programming of the Correlation Font Compiler

Nemo—GUI of the Correlation Font Compiler

Romain Liévin, Tim Singer—Ti-83+ Character Table

Rebma Boss—Idea for Word Wrap Mode

AND SPECIAL THANKS TO THE TESTERS:

COMING SOON!

Screenshot Credits

COMING SOON!

List of Correlation Commands

COMMAND	WHAT IT DOES
1:Asm(pgrmCORELATE	Lets you use Correlation with your Ti-Basic Program
0:Asm(pgrmCORELATE	Turns off Correlation so that your calculator will run normally
int("MYFONTRT	Tells Correlation to display text using the font found in pgrmMYFONTRT. This command works with any font compiled as a program with an 8-character name
ln(X Position, Y Position, String)	Displays text using a custom 6x8 font. This command works identically to Output(.
e^(Y Position, X Position, String)	Displays text using a custom font of any size. This command words identically to Text(.
int("DD	Updates the screen, displaying all text that you drew
int("RY	Turns on the Run Indicator
int("RN	Turns off the Run Indicator
abs(String, Start Position, Number Of Characters to Select	Returns a substring of the string you specified. This command works identically to sub(.
int("WY	Tells Correlation that Ws and Ms are bigger than other letters in your font
int("WN	Tells Correlation that all letters in your font are the same side

List of Advanced Correlation Commands

int(“M0	Turn on Clip Mode, so that text stops displaying once the edge of the calculator’s screen or window is reached
int(“M1	Turns on Wrap Mode, so that text moves to the next line once the edge of the calculator’s screen or window is reached
int(“M2	Turns on Word Wrap Mode
int(“M3	Turns on Map Mode
int(“FF	Tells Correlation that you specified a new custom window or a new map width for drawing text
int(“E0	Draws text using OVERWRITE
int(“E1	Draws text using OR
int(“E2	Draws text using XOR
int(“E3	Draws text using AND
int(“E4	Draws text using MASKED
int(“00 – 99	Selects a font from a font collection
int(“AY	Turns on Animated Text
int(“AN	Turns off Animated Text
det({Number Of Frames for First Animation, Number of Frames for Second Animation…}	Tells Correlation how many frames are needed for each animation you create

int(“CN	Turns Correlation Off, allowing you to use another asm library/program such as Xlib, AppVar or Celtic III
ZoomSto	Turns Correlation back on after using another asm program or library
int(“EY	Tells Correlation to display text immediately
int(“EN	Tells Correlation to avoid display any text drawn until it encounters the line int(“DD
int(“BY	Tells Correlation to erase a previously drawn character of text when you are ready to draw the same character again. This makes it look like your character is moving.
int(“BN	Tells Correlation not to erase any text that was drawn