brief contents

PART 1  GETTING STARTED WITH PROGRAMMING .......................... 1
  1  ■  Diving into calculator programming  3
  2  ■  Communication: basic input and output  25
  3  ■  Conditionals and Boolean logic  55
  4  ■  Control structures  76
  5  ■  Theory interlude: problem solving and debugging  107

PART 2  BECOMING A TI-BASIC MASTER ................................. 133
  6  ■  Advanced input and events  135
  7  ■  Pixels and the graphscreen  167
  8  ■  Graphs, shapes, and points  184
  9  ■  Manipulating numbers and data types  205

PART 3  ADVANCED CONCEPTS; WHAT’S NEXT ............................ 225
  10  ■  Optimizing TI-BASIC programs  227
  11  ■  Using hybrid TI-BASIC libraries  243
  12  ■  Introducing z80 assembly  260
  13  ■  Now what? Expanding your programming horizons  282
In late 2012, the members of my website discovered that Texas Instruments was testing a new color-screen version of the decade-old TI-83 Plus/TI-84 Plus series. This new calculator, the TI-84 Plus C Silver Edition, has the same processor, RAM, and operating system as the older monochrome calculators. It includes a new color LCD screen and rechargeable battery. In 2015, it was joined by the thinner, lighter TI-84 Plus CE, which packs a faster processor and more memory. Luckily for TI-BASIC programmers, the TI-BASIC (and z80 assembly) programming skills taught throughout Programming the TI-83 Plus/TI-84 Plus remain largely relevant to the new calculator. All of the commands and programming techniques you’ve learned in this book still apply, and the calculator still has a homescreen and graphscreen for text-based programs and graphical programs respectively. The bigger screen and the color features of the new calculators do require some extra knowledge to use properly.

In this appendix, you’ll discover what you need to know to translate the lessons of this book to the TI-84 Plus C Silver Edition and TI-84 Plus CE calculators. We’ll start with a high-level view of what’s new with these color calculators, which aspects resemble what you’re familiar with from programming monochrome calculators, and what new features you’ll need to be aware of. There are two main classes of new commands and skills you’ll need pertaining to the homescreen and the graphscreen respectively, and we’ll look at each of those topics in turn. Finally, the new color features require a new set of hybrid BASIC libraries, and that will be the last topic we cover in this appendix. Throughout, you’ll have plenty of sample programs
APPENDIX D  Programming the TI-84 Plus C Silver Edition and TI-84 Plus CE

and exercises to work with, many of which are programs you’ve seen elsewhere in this book but modified for the color calculator.

Let’s get started with the new features of the TI-84 Plus C Silver Edition and TI-84 Plus CE that you need to know as a user and as a programmer.

**D.1 The TI-84 Plus C Silver Edition and TI-84 Plus CE: what’s new?**

On the surface, the TI-84 Plus C Silver Edition and TI-84 Plus CE (shown in figure D.1) look like a mix of old and new. Although the buttons are the same as the older TI-84 Plus calculators, the TI-84 Plus CE has a slimmer case; turning either device on reveals a full-color screen with a new user interface, shown in figure D.2. Inside, the calculators are similar to the TI-84 Plus. Both have a larger Archive memory, the obvious color screen, and a rechargeable battery; the TI-84 Plus CE also has more memory and a faster processor. The software is essentially the same; TI-83 Plus and TI-84 Plus users find the math features in the TI-84 Plus C Silver Edition (or TI-84+CSE) and the TI-84 Plus CE (or TI-84+CE) familiar. Importantly, almost everything you learned about TI-BASIC in *Programming the TI-83 Plus/TI-84 Plus* can be used with the TI-84+CSE and the TI-84+CE.

![Figure D.1](image-url) The new features of the TI-84 Plus C Silver Edition (left) and TI-84 Plus CE (right) graphing calculators. Both have the new battery and screen. The TI-84 Plus C Silver Edition adds more Archive memory over the monochrome TI-84 Plus, whereas the TI-84 Plus CE also adds more RAM and a faster processor.
The TI-84+CE and TI-84 Plus CE operating systems are based heavily on the older calculators’ OS. Of course, almost every menu and feature had to be reformatted to fit the bigger screen. At the top of the screen in figure D.2, the new always-present mode toolbar takes up some of that expansive real estate. It shows the angle mode, graphing mode, complex number settings, and battery status, while the remainder of the screen holds a larger version of the classic homescreen, graphscreen, menus, or tools.

**D.1.1 New hardware and OS considerations**

The TI-84+CSE and TI-84+CE have large color screens, both add more Archive memory, and the TI-84+CE increases the RAM and processor speed. TI has modified the calculator’s operating system to take advantage of the larger screen, but the rest of the operating system, including what math features it offers, has changed little. Therefore, none of the features previously available to you as a programmer have disappeared, but on the other hand, your programs have few new capabilities available to them. For example, you can now draw shapes, text, and lines on the graphscreen in up to fifteen different colors, but unless you use Hybrid BASIC via Doors CSE 8, you can display text on the homescreen only in black. In this section, you’ll learn about the hardware changes and the new OS features with which your programs will interact.

**Hardware and the Homescreen**

Most prominently, the screen is now a 320 x 240-pixel color LCD. As a TI-BASIC programmer, you can access at most 15 of the 65,536 colors it can display; if you use Hybrid BASIC or assembly, you can use all of the colors. The screen is backlit, with an
adjustable backlight intensity, but TI-BASIC programs can’t manipulate the brightness. On the TI-84+CSE, there’s about 1KB less RAM available (23K total); on the TI-84+CE, you have up to 154KB of RAM. The TI-84+CSE and TI-84+CE have 4MB of Flash or Archive memory (with 3.0-3.5MB available to the user), more than twice as much as the TI-84+SE. In practical terms, this means that your TI-BASIC programs can now take advantage of larger variables holding numbers, lists, matrices, sprites, and graphics, stored in the Archive of the TI-84+CSE or in the RAM and Archive of the TI-84+CE.

The calculator’s homescreen is now 26 columns wide and 10 rows tall, expanded from 16 columns by 8 rows on the monochrome calculators. As before, each position can hold exactly one character, and TI-BASIC programmers don’t have access to MathPrint display features. Consequently, every TI-84+CSE and TI-84+CE program will be running on a MathPrint-capable operating system so games that use the homescreen should consider using the CLASSIC command at the beginning to ensure that they run at full speed. Unfortunately, you can print only in black text on a white background on the homescreen in pure TI-BASIC. Section D.4 will introduce how you can use Hybrid BASIC commands to draw homescreen text using Disp and Output with any possible background and foreground color.

**GRAPHSCREEN AND IMAGES**

The calculator’s graphscreen is the area that has changed most drastically from the older monochrome TI-83+ and TI-84+ calculators. Whereas the old graphscreen was 95 x 63 pixels and took up the full screen, the new graphscreen is 265 x 165 but takes a much smaller portion of the screen. The mode or status bar is still present at the top of the screen, and a thick border surrounds the graph area, both shown in figure D.3. As a programmer, you can control the color of the border, but you can’t remove it, just as you can’t hide the status area. Graphed functions can be drawn in any of 15 possible colors, and similarly, lines, points, and shapes can be drawn in the same colors. As with the monochrome calculators, you can control the axes, grid, graphed functions, and other graphing features to display and manipulate graphs from within your programs.

In this book, you learned about matrices, lists, programs, pictures, and the other types of data that your calculator can store. All of these are still available to TI-84+CSE users and programmers, and one new type called an Image has been added; Images are full-color and 133 x 83 pixels. As a user or a programmer, you can now change the background of the graphscreen to a solid color other than white, and you can also set it to one of these Images. Your calculator likely came with five images already loaded,
and you can use tools like SourceCoder 3 (http://sc.cemetech.net) or TI-Connect to make your own. In addition, pictures have changed from 95 x 63 pixels to 265 x 165 pixels, the size of the graphscreen; they also can use all 15 colors available to the drawing commands (see figure D.9).

**PICTURES VERSUS IMAGES** Pictures on the TI-84+CSE/TI-84+CE are 265 x 165 pixels but can only contain 15 possible colors, whereas Images are 133 x 83 pixels and can use any of 65,000 colors. You can only create Images on a computer using tools like SourceCoder and TI-Connect, but you can draw and store Pictures directly on the TI-84+CSE/TI-84+CE.

Now that you know all of these new features are available, I’ll show you how to interact with them using existing commands that take new arguments as well as using brand-new commands.

### D.1.2 New commands, new arguments

In addition to all the TI-83+/TI-84+ programming commands you already know you’ll need to learn a few new commands to take full advantage of TI-BASIC programming on the TI-84+CSE and TI-84+CE. You’ll also need to learn new arguments for existing commands like `Line()`, `Circle()`, and `Pxl-On()`. This section tabulates the new commands that you’ll need to know; sections D.2 and D.3 demonstrate using the new commands in programs.

All of the new commands presented in table D.1 are related to graphing or drawing. They allow you to control the background color or image and border color of the graphscreen, the colors of text and graphed functions, and how asymptotes like $y=1/x$ at $x=0$ are handled. Fortunately, there are no new programming commands; you don’t need to learn new ways to structure your programs or control the flow of execution. This nuance is a double-edged sword: older programs will be easy to port to the color calculators, but they don’t offer any powerful new programming features to explore. Most of these functions are self-explanatory. Notice that rather than using a `RecallImage` equivalent of `RecallPic` to render Image variables to the graphscreen, you use `BackgroundOn`. There is no way to store new Images on the calculator (see section D.1.3).

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackgroundOff</td>
<td>Sets the background of the graphscreen to white.</td>
<td>:BackgroundOff</td>
</tr>
<tr>
<td>BackgroundOn</td>
<td>Sets the background of the graphscreen to a color or an image. Colors and Image names are in the <code>VARS</code> menu. This is also the correct way to recall Images; there is no <code>RecallImage</code> command. BackgroundOn <code>&lt;COLOR&gt;</code> BackgroundOn <code>&lt;IMAGE&gt;</code></td>
<td>:BackgroundOn RED :BackgroundOn Image5</td>
</tr>
</tbody>
</table>
While some of the color calculators’ differences are in the existence of these new functions, others are in new arguments to functions that already existed on the TI-83+ and TI-84+ models. These changes apply solely to drawing and graphing functions, as required by the new LCDs’ higher resolution and ability to display in color. The commands that take new arguments, as well as a description of the new optional arguments, are listed in table D.2. There are additional commands that take a color as a final optional argument, namely \texttt{Tangent()}, \texttt{DrawF}, \texttt{Shade()}, and \texttt{DrawInv}. These aren’t covered here, as they aren’t frequently used in programs. For their arguments, go to the \texttt{2nd PRGM} (DRAW) menu, find the function in question, and press \texttt{}. The \texttt{Pt-} commands now work with 2x2 squares of pixels unless you specify the optional \texttt{MARK} argument. If you have a TI-84+CSE, see the note on “TI-OS 4.x Color Point Bug”.

\textbf{TI-OS 4.X COLOR POINT BUG} In TI-OS 4.0 and 4.2 for the TI-84+CSE, using the \texttt{MARK} and \texttt{COLOR} arguments together for \texttt{Pt-On()} and \texttt{Pt-Change()} doesn’t work. The points drawn will appear in black until the program finishes, when they might change to their proper color. Unfortunately, you can’t therefore use the point commands \texttt{MARK} argument in interactive programs when you also want colors other than black. The TI-84 Plus CE doesn’t suffer from this bug.

You’ll get a chance to see these new commands in action in homescreen and graph-screen programs in section D.2 and D.3, but first you need to know what programs and tools are available to make it easier to program for the TI-84 Plus C Silver Edition.
D.1.3 New tools and apps

“Appendix C” of Programming the TI-83 Plus/TI-84 Plus presented a list of handy resources, including emulators, editors, image converters, and forums. Most of these are still applicable, but I’d like to highlight a few specific resources that are particularly helpful for the TI-84+CSE:

- **Emulators:** Both jSTIfied (http://www.cemetech.net/projects/jstified) and WabbitEmu (http://wabbit.codeplex.com) currently support the TI-84+CSE. At press
time, neither supports the TI-84+CE, but jSTified is slated to get support. TI’s official SmartView classroom tool also emulates both calculators.

- **IDEs:** SourceCoder 3 (http://sc.cemetech.net) has a host of tools for editing and creating TI-84+CSE and TI-84+CE programs, converting Picture and Image variables, and working with other data types like lists and matrices. TokenIDE (http://cemete.ch/DL515) has invaluable tools for working with xLIBC sprites and images.

- **Shells:** Doors CS 7 for the monochrome calculators has a sibling called Doors CSE 8 for the TI-84+CSE. It offers many of the features that Doors CS gives TI-BASIC and assembly programmers on the monochrome calculators. Indeed, the Hybrid BASIC features discussed in section D.4 require Doors CSE 8.1 or higher. A Doors CSE port for the TI-84+CE is planned but not yet available as of press time.

- **Communities:** Cemetech has adopted the TI-84+CSE and TI-84+CE enthusiastically, producing programs, games, shells, tools, and educational material for the calculator. The site can be found at http://www.cemetech.net, and the forum is at http://www.cemetech.net/forum/.

Now that you know what new commands are available, what other commands have new arguments, and where you can find the resources you need, let’s dive into how you can use these in your own programs. We’ll start with a few simple examples of homescreen programs reformatted for the larger screen, then move onto graph-screen programs that use colored drawing commands and Hybrid BASIC programs that have even more control over text and graphics on the screen.

### D.2 Homescreen programs

Throughout this book, we have worked together on programs that take advantage of the homescreen for input and output using `Input`, `Prompt`, `Disp`, `Output()`, and `getKey`. Most of these can be carried over to the TI-84 Plus C Silver Edition and TI-84 Plus CE with few changes. In many cases the only important difference is the larger size of the homescreen, now 10 rows by 26 columns instead of 8 rows by 16 columns. This section will demonstrate how two of the examples you’ve seen elsewhere in this book can be reformatted to fit the larger homescreen.

We’ll start by looking at the MOVECHAR program from chapter 6, which let the user move an M around the homescreen. The only necessary change to make this program work on the TI-84+CSE and TI-84+CE is to modify where it thinks the bottom and right edges of the homescreen lie. Our second program is a simple Snake game, where you control a line of characters roaming the homescreen. This program has equally few specializations to make it work on the TI-84+CSE and TI-84+CE; a clever programmer could easily modify it to work on the monochrome TI-83+ and TI-84+ calculators. Together, these two programs will demonstrate how easy it is to apply your existing knowledge of the homescreen to the TI-84+CSE and TI-84+CE.

Let’s dive right in with the first program, the MOVECHAR program from section 6.2.1 modified for the TI-84+CSE and TI-84+CE.
**D.2.1 Movement and the TI-84+CSE or TI-84+CE homescreen**

In chapter 6, you learned about event loops that continually check the keypad and react to keys pressed. One of the programs demonstrating this concept let you move the letter M around the homescreen with the arrow keys, then press [CLEAR] to exit the program. If you need to refresh your memory, you should review section 6.2.1 for the full explanation of how this program works, including the concept of event loops and `getKey`. This program was one of our simplest event loop examples, on top of which we built the Mouse and Cheese game later in chapter 6.

So what made that MOVECHAR program specific to the TI-83+ and TI-84+ calculators, and what must be changed to convert it to MOVECHRC for the TI-84+CSE or TI-84+CE? The answer to both questions, happily, is not much at all. In fact, if you send MOVECHAR from chapter 6 to your TI-84+CSE or TI-84+CE, it will run with no error messages. The only problem will be that you can’t move the M to the right and bottom edges of the screen, which provides the clue to what we need to change. On the monochrome calculators, the homescreen is 16 characters wide and 8 characters tall. Consequently, MOVECHAR prevents the user from moving the M past column 16 or row 8. Because the TI-84+CSE/TI-84+CE homescreen has 26 columns and 10 rows, the limits need to be changed. A screenshot of a form of MOVECHAR modified thus to work properly on the color calculators is shown in figure D.4.

The full source code for prgmMOVECHRC is provided in listing D.1. The 16 lines of code are nearly identical to the original MOVECHAR; only 3 lines have been changed. These changes are all related to the new homescreen dimensions:

1. The initial coordinates are now A=13 (column) and B=5 (row). This is roughly the center of the 26-column, 10-row homescreen.
2. To make sure the M does not go past the right edge of the screen, the conditional for the right arrow key has been changed from `If K=26 and A<16` to `If K=26 and A<26`. Now, the user can only move the M to the right if it is to the left of the last column.
3. Similarly, the new conditional for the down arrow key is `If K=34 and B<10`. If the M is in the last row, corresponding to B=10, then the condition B<10 is false, and `B+1` will not be executed.

Try running this program on your own calculator, and you’ll see that now the M can reach all four edges of the homescreen.
PROGRAM: MOVECHRC
:13→A:5→B
:ClrHome
:Repeat K=45
:Output(B,A,"M"
:getKey→K
:If K
:Output(B,A,"
:If K=24 and A>1
:A-1→A
:If K=26 and A<26
:A+1→A
:If K=25 and B>1
:B-1→B
:If K=34 and B<10
:B+1→B
:End

Moving a single character around the screen is a nice proof-of-concept, and I challenge you to take this lesson and apply it to making the Mouse and Cheese game work on the TI-84+CSE and TI-84+CE. To keep things interesting, let’s explore further with homescreen programs and the TI-84+CSE/TI-84+CE with a new game, Snake.

D.2.2 Snake game on the TI-84+CSE and TI-84+CE

Variations on Snake are perennial favorites among casual gamers, giving the popular Tetris a proverbial run for its money in the number of different versions for different platforms available. TI graphing calculators are no exception: many a curious student has picked up TI-BASIC or assembly and tried their hand at creating a Snake clone. We’ll join those proud ranks with a simple Snake game of our own. It has only a moving, fixed-length snake, with no walls, food, or scoring. I encourage you to add all those elements and more yourself and release the finished product as your own.

In case you have somehow avoided all Snake games in the past, the game consists of a line (the snake) that you pilot around a play field. It moves continuously, and you must use the arrow keys to change the direction the snake faces as it moves. In most Snake games, your objective is to eat food scattered around the play field while avoiding your growing tail, the walls, and in some cases poison or enemies. Our particular version will be simple, implementing only the snake. I will first show you the core concept of fast Snake games, a data structure called a circular buffer, then present the game itself and explain how it works.

Prerequisite: Circular Buffers

In chapter 9, you learned about data structures like lists and matrices. Whereas real variables like \(A\) and \(X\) store single numbers, lists store a sequence of numbers, and
matrices contain a two-dimensional array of numbers. There are many data structures in programming and computer science that store more complex sets of data. In this section we’ll look at a structure called a circular buffer. Circular buffers are similar to lists in that they store a one-dimensional, ordered set of numbers. The difference between a list and a circular buffer is that the latter also holds one or two pointers to specific elements. The most general type of circular buffer, shown in figure D.5, contains a set of data, a pointer to the first item, and a pointer to the last item. It is called a buffer because you can remove items by reading them and moving the head, or add items by writing them and moving the tail. They can therefore be used to temporarily hold data until a program is ready to use it, even if new data is continually arriving.

We’ll use a pair of simplified circular buffers for our Snake game to hold the coordinates of each of the segments of the snake’s body. As shown in figure D.6, one circular buffer, with elements held in list \( L_1 \), will contain the \( x \) coordinates of all the segments. The other, with elements stored in \( L_2 \), will hold the \( y \) coordinates. The buffers will be simplified by maintaining a single pointer shared across both buffers, pointing to the single element that is both the head and the tail of the buffer. In a regular circular buffer, the number of items between the head and the tail indicates how many elements it currently holds. Because the Snake game uses the same pointer for the head and tail of both buffers, both buffers are considered to be completely full at all times.

Every time the snake moves, we first erase the segment specified by the \( x \) and \( y \) coordinates corresponding to the elements of \( L_1 \) and \( L_2 \) that the head/tail pointer indicates. Next, we draw the snake’s new head, and store the coordinates of that head into the same list elements where the discarded, erased tail segment had been stored. Finally, we move the head/tail pointer forward one element, ready to erase the snake’s new last segment.
THE SNAKE GAME

Now we can combine the circular buffer principles together with what you already know about event loops and text on the homescreen to create the rudimentary Snake game. The code is presented in listing D.2. Following the usual structure for this sort of program established in chapter 6, it first sets up the necessary variables and data storage, then enters a main game loop that repeats until the player presses \texttt{CLEAR}. Inside the loop, it erases the tail, updates the circular buffer, draws the head, and then accepts key input from the user. It also handles what happens when the snake runs off one of the edges of the screen, making it reappear on the opposite edge. You can see a screenshot from this game in action in figure D.7.

**Listing D.2 A simple homescreen Snake game**

```
PROGRAM:SNAKE
ClrHome
13→A:5→B:10→X:1→P:26→D
X→dim L1
X→dim L2
Fill(B,L1
Fill(A,L2
Repeat K=45
Output(L1(P),L2(P),"[one space]
Output(B,A,"0
B→L1(P):A→L2(P
P+1→X(P=X→P
getKey→K
If max(K={24,25,26,34
K→D
A+(D=26)-(D=24→A
B+(D=34)-(D=25→B
A+26((A=0)-(A=27→A
B+10((B=0)-(B=11→B
End
```

In this program, we use (A,B) as the (x,y) coordinates of the head, X as the buffer length, P as the head/tail pointer, and D as the snake’s current direction (26 corresponds to the right arrow key). We fill both lists of coordinates with the coordinates of the head so that the snake will grow to full length when the game starts. There are two sections of code here that might be confusing to the intermediate TI-BASIC programmer. First, this section of code updates the circular buffer:

```
B→L1(P):A→L2(P
P+1→X(P=X→P
```
It stores the x coordinate A into element P of L1 (the element where the head/tail pointer P points), and the y coordinate B into the corresponding element of L2. The code then increments P by 1 so that it points to the next element of both lists. If P was equal to X (the length of the lists and hence the circular buffers), then P+1 would be off the end of the two lists. Therefore, P+1-X(P=X)=P+1-X(0)=P+1 if P<X, or P+1-X(P=X)=P+1-X(1)=X+1-X=1 if P=X. You could make this code more readable but slower by incrementing P, then adding a conditional to check if P>X and set P back to 1 if so.

The second potentially confusing section of code deals with wrapping the snake’s head around if it reaches the edge of the screen. These two lines operate similarly to the code discussed above:

A+26 ((A=0) - (A=27) - A
B+10 ((B=0) - (B=11) - B

In this case, A is set to A+26=26 if A is equal to 0, and A-26=1 if A is 27. If the snake’s head goes off the left or right edge, it reappears on the opposite edge. Similarly, the second line moves B to the first or last row if it passes the bottom or top edge of the homescreen, respectively.

**GOING FURTHER**

This Snake demo is a nice start to a game, but lacks a great deal in the way of gameplay and features. I challenge you to discern first how to scatter a random piece of food on the playfield, then how to expand the circular buffers when the snake eats a piece of food. You could start with buffers with extra capacity and only occasionally resize them, or you could come up with a way to resize the buffers every time the snake expands. You could then consider adding collision detection, so that the player will lose if he or she hits the snake’s body or walls (hint: look to MATRXRPG in chapter 9 for inspiration on this).

The new TI-84+CSE/TI-84+CE homescreen doesn’t offer many new challenges, but the TI-84+CSE/TI-84+CE graphscreen is sufficiently different to require learning a few new techniques.

**D.3 Programs, pixels, and the color graphscreen**

In section D.1.1, you read about the expanded dimensions of the graphscreen and its ability to render drawing commands, Pictures, and Images in color. Section D.1.2 presented the specific drawing commands used to control the TI-84+CSE or TI-84+CE graphscreen. Now, we’ll get a chance to put that knowledge together into a few small programs. First, I’ll challenge you to create a program that can display a string of text, rendered in random colors, at random locations on the graphscreen. Second, I’ll walk you through modifying the painting program from chapter 7.4 for the color calculator. Instead of using ENTER to lift and drop a pen, though, you’ll use ENTER to cycle between the available ink colors.

You could probably modify some of your existing monochrome graphscreen programs to the TI-84+CSE and TI-84+CE by adjusting how big the programs think the
graphscreen is, as we did for the MOVECHAR program in section D.2.1. Instead, let’s start with a more complex, slightly more fun variation: combining the mechanical difference with the commands to draw colored text on the graphscreen.

**D.3.1 Example: Random colored text**

In chapter 7, you learned about drawing strings of text on the graphscreen with the `Text(` command. Given pixel coordinates and a string, it would draw the string to the graphscreen at those coordinates. The `Text(` command on the TI-84+CSE and TI-84+CE is identical to the command on the monochrome calculators, and doesn’t accept any new arguments to specify the color of the text drawn. However, there’s a new `TextColor(` command. Once you use this command to set a text color, all subsequent `Text(` commands will draw in that color until you specify a new color. We’ll use these two commands to randomly draw "HELLO"s around the graphscreen in random colors, as in figure D.8.

To understand how we can efficiently pick a random color, we first need to look at the numbers that represent each color. What? Look at figure D.9, and you’ll see what I mean. Even though all of the colors in the menu have a token like RED, BLUE, or GREEN, you can also refer to them with a number like 11, 10, and 14, respectively. Notice also that some text has a gray background; unfortunately, TI-BASIC programs can’t control this text background, which is added by the OS. Because the colors can be reference by the numbers 10-24, we can pick a random color by passing a random number between 10 and 24 to `TextColor(`. As I explained in section 9.4, the `randInt(` command can be used to generate a random integer. Therefore, this line of code will set the stored text color to a random color among the calculator’s 15 possibilities:

```
:TextColor(randInt(10,24)
```

Now that we have a random text color, we want to display "HELLO" at a random position on the screen. The top-left corner of the graphscreen has pixel coordinates (row,
column) = (0,0), and the bottom-right corner has coordinates (row, column) = (164, 264). Each line of text is 14 pixels tall, and assuming each character is about 8 pixels wide, we can assume that our "HELLO" will be roughly 40 pixels wide. Adding a few pixels to make sure we don’t run off the edge, this means we can place our text anywhere from row 0 to row 150, and from column 0 to column 222 to have it fit on the screen. As you might expect, we’ll use \( \text{randInt(0,150)} \) twice more to select the row and column for the text. \( \text{randInt(0,150)} \) will provide the row, and \( \text{randInt(0,222)} \) will give the row. Here’s the finished \( \text{Text()} \) command with \( \text{TextColor()} \):

\[
:\text{TextColor(randInt(10,24)}
:\text{Text(randInt(0,150),randInt(0,222),"HELLO
\]

We want to repeat this until the user presses a key, and we probably want to clear the screen before we begin. We can wrap the code in a simple \( \text{Repeat} \) loop and add a \( \text{ClrDraw} \):

\[
:\text{ClrDraw}
:\text{Repeat getKey}
:\text{TextColor(randInt(10,24)}
:\text{Text(randInt(0,150),randInt(0,222),"HELLO}
:\text{End}
\]

Finally, let’s add code to turn off graphed functions, the grid, the axes, and any background before the program starts, and then restore all the user’s settings when the program ends. You should consider using this prologue and epilogue code in all of your graphscreen programs to avoid trampling the user’s graphing settings. The finished program is shown in listing D.3.

**Listing D.3**  Drawing random colored text with RNDCOLOR

```
PROGRAM:RNDCOLOR
:StoreGDB 0
:BackgroundOff:AxesOff
:GridOff:FnOff
:ClrDraw
:Repeat getKey
:TextColor(randInt(10,24)
:Text(randInt(0,150),randInt(0,222),"HELLO
:End
:RecallGDB 0
:ClrHome
```

Prologue: preserve graph settings

Epilogue: restore original graph settings

Give prgmRNDCOLOR a try on your calculator; see if you can make it do new, interesting things. Make sure you also understand the prologue and epilogue marked in listing D.3, as these would be good additions to any of your own TI-84+CSE programs that use the graphscreen. Once you feel comfortable with the new skills that RNDCOLOR teaches, let’s move on to a modification of the painting program from chapter 7.
D.3.2 **Exercise: Painting in color**

In chapter 7, you were busy learning about writing programs that could use the graphscreen. Along the way, you combined the event loop lessons from chapter 6 with your newfound knowledge of the graphscreen to create a painting program. With this program, you could use the arrow keys to move a pen around the screen, drop it with ENTER to draw as you moved the pen, then lift it again with ENTER. In this section, we’ll make a few simple modifications to this program for the TI-84+CSE and TI-84+CE. Specifically, we’ll change it so that:

- It knows about the new size of the graphscreen in pixels.
- It has a prologue and epilogue matching the ones in prgmRNDCOLOR above.
- Pressing ENTER changes the pen color instead of lifting and dropping the pen.

When we finish, the output of using this painting program might resemble figure D.10

If you take a look at the original program’s code in listing 7.4 (reproduced in listing D.4), you’ll see that it has a lot in common with the MOVECHRC program in listing D.1. It also tracks the coordinates of an object moved by the arrow keys. Instead of waiting until the user presses a key to erase the moving item (the pixel being painted), it continuously draws and erases it to provide a grayscale effect. We’ll keep this effect, but all of our Pxl-On commands will need the current color. The old PAINT program used variable P to store whether the pen was drawing (P=1) or lifted off the “paper” (P=0). We’ll repurpose P to store the current color; as with the RNDCOLOR program, it will hold a value between 10 and 24, corresponding to the fifteen colors shown in figure D.9.

**Listing D.4** A monochrome painting program, reproduced from chapter 7

```plaintext
PROGRAM:PAINT
:AxesOff:ClrDraw
:48→A:32→B
:0→P
:Repeat K=45
:Repeat K
:Pxl-Change(B,A
:getKey→K
:Pxl-Change(B,A
:End
:If K=24 and A>0
:A-1→A
:If K=26 and A<94
:A+1→A
```

![Figure D.10 Testing the modified PAINTC program on the TI-84+CSE or TI-84+CE](image)
Now we have the different pieces we need to plan. We’ll first change the starting point to the middle of the 265-column, 165-row color graphscreen, and then write the conditional statements that make the pen accommodate the new right and bottom edge positions. We’ll then add the prologue and epilogue from prgmRNDCOLOR, using GDB0 to store and recall the state of the graphscreen rather than simply toggling the axes off and on. Finally, we’ll make \texttt{ENTER} increment \texttt{P} to the next color in the set between 10 and 24, and change the \texttt{Pxl-On(} statements to draw in the color \texttt{P} holds. Listing D.5 shows how the final program could be written. I encourage you to try to make the modifications yourself first, without peeking at listing D.5.

\begin{verbatim}
PROGRAM:PAINTC
:StoreGDB 0
:BackgroundOff:AxesOff
:GridOff:FnOff
:ClrDraw
:132→A:83→B
:10→P
:Repeat K=45
:Repeat K
:Pxl-Off(B,A
:getKey→K
:Pxl-On(B,A,P
:End
:If K=24 and A>0
:A-1→A
:If K=26 and A<264
:A+1→A
:If K=25 and B>0
:B-1→B
:If K=34 and B<164
:B+1→B
:If K=105
:P+1→P
:If P=25
:10→P
:End
:RecallGDB 0
:ClrHome
\end{verbatim}
**GOING FURTHER WITH PAINTC**

There are myriad ways you could change and expand this painting program. For example, you could make it indicate in some way what the current color is by adding a border around the edges of the graphscreen with `Line(` commands in the current color. You could add a way to lift and drop the pen in addition to changing the color. You could add a menu that lets the user store their sketches as Picture variables or recall an existing Picture to draw on. You could even give the user the ability to set the thickness of the line.

I’ll leave you to explore those avenues on your own. When you’re ready, proceed to the last subject of this chapter, Hybrid BASIC on the TI-84+CSE.

### D.4 Hybrid BASIC and Doors CSE

In chapter 11, you learned about Hybrid BASIC on the TI-83+ and TI-84+ calculators. To refresh your memory, Hybrid BASIC is the term for combining TI-BASIC with extra functions provided by Apps or assembly programs like Doors CS, xLIB, and Celtic III. Hybrid BASIC libraries are available for the TI-84+CSE via Doors CSE 8; at press time, no TI-84+CE are yet available, although they’re planned. If you have a TI-84+CE, this section does not apply to you. Doors CSE offers two sets of Hybrid BASIC libraries, Celtic II CSE and xLIBC. Both share attributes with their monochrome counterparts, but both have been heavily overhauled for the TI-84+CSE.

Celtic II CSE mostly provides functions for manipulating data and variables. You can use it to create, read, and modify programs and application variables in RAM and Archive. It also includes functions for drawing sprites and changing the color of text on the homescreen. xLIBC is for more complex graphics and sprites. In this section, I’ll briefly explain those two sets of Hybrid BASIC functions. Then, I’ll show you MTRXRPGC, a color modification of our homescreen RPG game from chapter 9. Let’s start with what you can do with the TI-84+CSE Hybrid BASIC libraries.

#### D.4.1 The color Hybrid BASIC libraries

Among the many lessons about Hybrid BASIC we discussed in chapter 11 were the header necessary for Hybrid BASIC programs, the functions available in the existing libraries, and a few example programs demonstrating what could be done with these libraries. As you might expect, Hybrid BASIC programs on the TI-84+CSE also require a special header, which ensures that the libraries are present on the user’s calculator. Although there are fewer libraries available for the TI-84+CSE than for the monochrome calculators, the available functions cover many things BASIC programmers might need, from sprites, to scrolling graphics, to file manipulation.

The header for TI-84+CSE Hybrid BASIC programs is shown here. Notice two particular elements: the 16 x 16-pixel icon on line 2, and the minimum Doors CSE version number encoded on line 3.

```
::DCS
:"Icon data...
:If 81>det([[20 16 x 16 icon, with one hex nibble per pixel; 256 characters total
\< 80 for DCSE 8.0, 81 for DCSE 8.1, 90 for DCSE 9.0, etc.
```
The latest version of Doors CSE is Doors CSE 8.1, so this header ensures that all users have Doors CSE 8.1 or later on their calculator. If Doors CSE isn’t on the calculator, then \[ \det([[20]]) \] is equal to 20, because the determinant of a single-element matrix is the value of the element; the error will then be displayed.

Before we get to an example, let me give you a fast, dense overview of the functions in Celtic II CSE and xLIBC. For the full details on all of these functions, you can find the Doors CSE documentation at http://dcs.cemetech.net/?title=SDK.

**Simple Hybrid BASIC: Celtic II CSE**

Celtic II CSE provides a total of 13 functions for manipulating files, drawing sprites, and changing the color of text on the homescreen. Every function is prefaced with the \[ \text{det(} \text{token, like Celtic III on the TI-83+ and TI-84+}. \] Functions to work with files include ReadLine, ReplaceLine, InsertLine, DeleteLine, CreateVar, DeleteVar, ArcUnarcVar, and VarStatus. SpecialChars lets you get \[ \rightarrow \text{and} \quad " \] in strings. BufSprite and BufSpriteSelect draw 16-color sprites from hex strings. DispColor lets you change the color used for \[ \text{Disp} \] and \[ \text{Output(}. \] Finally, ExecArcPrgm works similar to its monochrome counterpart in xLIB.

**Complex Hybrid BASIC: xLIBC**

To increase the speed of programs and decrease the size of sprites and graphics, xLIBC puts the 320 x 240-pixel LCD into a special half-resolution, 160 x 120-pixel mode with the xLIBCSetup routine. It sacrifices screen resolution for speed by making every pixel a 2 x 2 square of physical pixels. In this mode, it provides sprite and tile-map functions via DrawTile and DrawSprite. To make sprite drawing faster, you can cache sprites in RAM using ManagePic. DrawString lets you draw text on the screen, and DrawShape draws lines, rectangles, and pixels. Other available functions include a faster, more powerful GetKey and UpdateLCD.

Explaining all of these new functions would take nearly the space of a second book. Consult the Doors CSE SDK documentation, and follow along with this next exercise.

**D.4.2 Exercise: Output() in color with MTRXRPGC**

In chapter 9, you learned to make a matrix-based RPG game, fittingly called MTRXRPG. The game included a \( \pi \) that the player controlled; the object was to get to the exit (a °) without encountering the enemy, a \( T \). The enemy \( T \) tries to move towards the player’s \( \pi \), because touching the enemy injures the player. The game used an 8-row by 16-column matrix to hold the map, with one element corresponding to each character that can fit on the screen. A screenshot of that game is reproduced in figure D.11.

To modify this game for the TI-84 Plus C Silver Edition, the bigger homescreen requires a bigger matrix. Because the homescreen is now 10 rows by 26 columns, the
matrix must also be 10 x 26, and the map data stored in the matrix must be adjusted accordingly. The other major change that we'll make is to use color with the `Output` command. Unfortunately, the TI-OS on the TI-84+CSE does not offer any such feature, so we'll use Hybrid BASIC. The `det(12)` command in Doors CSE lets you specify a foreground and background color for all text printed to the homescreen with `Disp` and `Output` until you specify a new color.

**SAMPLE SOLUTION**

One possible solution to adapting MATRXRPG for the TI-84+CSE is shown in listing D.6. It uses `det(12)` to change the color for `Output`, and expands the matrix that stores the map to be the same size as the expanded TI-84+CSE homescreen. This is only one possible solution, and a creative programmer could think of many other ways to adapt this game for the TI-84+CSE. For example, you could use the Celtic II CSE sprite routines to draw the map or experiment with the xLIBC tile mapping commands.

**Listing D.6 The MTRXRPGC game**

```plaintext
PROGRAM: MTRXRPGC

:DCS
"33BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
Open a single space in the middle wall

Add left and right walls to the map

Add goal/exit (4 represents the exit)

1 ‐>[C](6,13
For(X,1,10
2 ‐>[C](X,1
2 ‐>[C](X,26
End
4 ‐>[C](9,25
Lbl MM
Lbl Q:ClrHome
Return
Lbl H
Pause "HELP GOES HERE
Goto MM
Lbl A
100→H:2→A:2→B
24→M:9→N
ClrHome
For(D,1,10
For(C,1,26
[C](D,C
det(12,159(Ans=4),28(Ans=4),229(Ans=1 or Ans=4),175(Ans=1 or Ans=4
Output(D,C,sub(" O+°",Ans,1
End:End
det(12,0,0,229,175
Repeat K=45 or H≤0 or 4=[C](B,A
Output(B,A,"π
Output(N,M,"
Repeat K or H≤0
If rand<0.2
Then
Output(N,M,
If rand>0.5
Then
If A<M and 2≠[C](N,M−1
M−1→M
If A>M and 2≠[C](N,M+1
M+1→M
Else
If B<N and 2≠[C](N−1,M
N−1→N
If B>N and 2≠[C](N+1,M
N+1→N
End
Output(N,M,"
If N=B and M=A
H−30→H
End
getKey→K:End
Output(B,A,"[one space]"
If K=24 and 2≠[C](B,A−1
A−1→A
If K=26 and 2≠[C](B,A+1
A+1→A

Choose color for spot: space (1); wall (2); exit (4)

Run loop until one of three game over conditions

The small transpose T can be found in the MATRIX menu

Move the enemy in 20% of the main loops

Nested loop draws the map

One single space to clear the sprite
Experimenting with the Hybrid BASIC functions is the best way to become familiar with how they work. The Doors CSE SDK offers a sample xLIBC game, and several Celtic II CSE and xLIBC educational programs and games are already available on ticalc.org, Cemetech, and in other calculator program archives. Looking at the source of those programs would be another great way to understand TI-84+CSE Hybrid BASIC.

D.5 Summary

This appendix presented a broad overview of TI-BASIC programming on the TI-84+CSE and TI-84+CE graphing calculators. Although there are a new functions, new arguments to existing commands, and new Hybrid BASIC libraries, programming is largely the same as for the TI-83+ and TI-84+. I first showed you the hardware and software differences that the new calculators introduce, then moved on to the new drawing commands and new arguments to existing drawing commands. We applied the requirements of the new hardware and software to a few example programs on the homescreen and graphscreen, including MOVECHRC, SNAKE, RNDCOLOR, and PAINTC. I talked briefly about the Hybrid BASIC libraries available on the TI-84+CSE, then showed you an example of the MATRXRPG program modified into a TI-84+CSE Hybrid BASIC game.

As always, the best thing you can do with your new knowledge is to explore, experiment, and expand on it. The material in this chapter is far from an exhaustive tutorial on everything the TI-84+CSE can do. If you haven’t done so already, I strongly encourage you to read the rest of Programming the TI-83 Plus/84 Plus to learn all about what your trusty graphing calculator can do. And as always, don’t hesitate to get in touch with me or visit the Cemetech forum with questions or to show off your own projects.
Programming the
TI-83 Plus/TI-84 Plus
Christopher R. Mitchell

The TI-83 Plus and TI-84 Plus are more than just powerful graphing calculators—they are the perfect place to start learning to program. The TI-BASIC language is built in, so you have everything you need to create your own math and science programs, utilities—even games.

Programming the TI-83 Plus/TI-84 Plus teaches universal programming concepts and makes it easy for students, teachers, and professionals to write programs for the world’s most popular graphing calculators. This friendly tutorial guides you concept-by-concept, immediately immersing you in your first programs. It introduces TI-BASIC and z80 assembly, teaches you tricks to slim down and speed up your programs, and gives you a solid conceptual base to explore other programming languages.

What’s Inside
• Works with all models of the TI-83, TI-83+, and TI-84+
• Learn to think like a programmer
• Learn concepts you can apply to any language
• Advanced concepts such as hybrid BASIC and ASM

This book is written for beginners—no programming background is assumed.

Christopher Mitchell is a PhD candidate and a recognized leader in the TI-83+/TI-84+ programming community. He hosts discussions and collaboration on calculator programs and projects at his website, Cemtech.

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit manning.com/ProgrammingtheTI-83Plus/TI-84Plus