

Bonus Chapter 6

Controlling Program Flow

In This Chapter

- ▶ Entering program control commands in your program
- ▶ Using decision commands (If, If . . . Then . . . End, If . . . Then . . . Else . . . End)
- ▶ Using looping commands (While . . . End, Repeat . . . End, For . . . End)
- ▶ Using branching commands (Goto, Menu)
- ▶ Stopping the execution of a program
- ▶ Pausing the execution of a program
- ▶ Using an external program as a subroutine in your program

The flow of a program is controlled by decision commands such as **If . . . Then . . . Else . . . End**, looping commands such as **For . . . End**, and branching commands such as **Goto**. Calling another program from within your program also controls the flow of a program. This chapter tells you how to use these and other commands that control the flow of your program.

Entering Control Commands in a Program

The Program Control menu, which houses the control commands, is available only when you're using the Program editor to create a new program or to edit an existing program. A picture of the Program Control menu appears in Figure B6-1. (Chapter B5 explains creating and editing programs.)

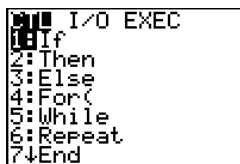


Figure B6-1: The Program Control menu.

To enter a control command in a program being written on the calculator, press **PRGM**, use **▼** to move the indicator to the desired control command, and then press **ENTER**. The command is then entered at the location of the cursor in the Program editor.

To enter a control command in a program being written on the computer using TI-Graph Link, click **PRGM** on the calculator keyboard to the left of the Program editor and then double-click the desired control command appearing in the right panel of the Program Control menu. The command is then entered at the location of the cursor in the Program editor. To access the calculator window again, close the Program menu.



When using TI-Graph Link to create a program, if you should accidentally close the calculator keyboard window, to get it back click **Window**⇒**TI-83 Plus** [or whatever the name of your calculator is] **Keypad**.

Using Decision Commands

The calculator can handle three decision commands (**If**, **If . . . Then . . . End**, and **If . . . Then . . . Else . . . End**). This section describes how to use them in a program.

The **If** command

The structure of the **If** command appears in the first picture in Figure B6-2. If the condition following the **If** command is true, the program executes the command following the **If** statement (command 1) and then moves on to the next command in the program (command 2). If the condition following the **If** command is false, the program skips the command following the **If** statement (command 1) and then moves on to the next command in the program (command 2).

An example of using the **If** command appears in the second picture in Figure B6-2. The program in this picture gives a 10 percent discount on items that cost \$50 or more. The input and output commands (Input, Disp) in this program are housed in the Program I/O menu which is accessed by pressing `[PRGM]`. Commands in this menu are explained in Chapter B7. You can enter the inequality that appears in this picture by pressing `[2nd][MATH][4]`.

<pre>PROGRAM:IF :If condition :command 1 :command 2</pre>	<pre>PROGRAM:DISCOUNT :Input "Price=", P :If P≥50 :.9P→P :Disp "Discount Price",P</pre>
Structure	Example

Figure B6-2: Using the **If** command.

The If . . . Then . . . End Command

The structure of the **If . . . Then . . . End** command appears in the first picture in Figure B6-3. If the condition following the **If** command is true, the program executes the commands between **Then** and **End** (commands 1) and then moves on to the next command in the program (command 2). If the condition following the **If** command is false, the program skips the commands between **Then** and **End** (commands 1) and then continues on to the next command in the program (command 2).

An example of using the **If . . . Then . . . End** command appears in the second picture in Figure B6-3. The program in this picture gives a 10 percent discount on items that cost \$50 or more and then takes off another \$10 if the discounted cost is over \$100.

<pre>PROGRAM:IFTHEN :If condition :Then :commands 1 :End :command 2</pre>	<pre>PROGRAM:DISPRICE :If P≥50 :Then :.9P→P :If P≥100 :P-10→P :End :Disp "Discount</pre>
Structure	Example

Figure B6-3: Using the **If . . . Then . . . End** command.

The If . . . Then . . . Else . . . End Command

The structure of the **If . . . Then . . . Else . . . End** command appears in the first picture in Figure B6-4. If the condition following the **If** command is true, the program executes the commands between **Then** and **Else** (commands 1), skips the commands between **Else** and **End** (commands 2), and then moves on to the next command in the program (command 3). If the condition following the **If** command is false, the program skips the commands between **Then** and **Else** (commands 1), executes the commands between **Else** and **End** (commands 2), and then moves on to the next command in the program (command 3).

An example of using the **If . . . Then . . . Else . . . End** command appears in the second picture in Figure B6-4. The program in this picture divides a number by 2 if it is even, or adds 3 to the number if it isn't.

<pre>PROGRAM: THENELSE : If condition : Then : commands 1 : Else : commands 2 : End : command 3</pre>	<pre>PROGRAM: DIV2 : Input "NUM=",N : ClrHome:Disp N : If fPart(N/2)=0 : Then:N/2→N : Else:N+3→N : End : Disp N</pre>
---	---

Structure

Example

Figure B6-4: Using the If . . . Then . . . Else . . . End command.

Using Looping Commands

The calculator can handle three looping commands (**While . . . End**, **Repeat . . . End**, and **For . . . End**). This section describes how to use them in a program.

The While . . . End command

The structure of the **While . . . End** command appears in the first picture in Figure B6-5. If the condition following the **While** command is true, the program executes the commands between **While** and

End (commands 1) and then returns to the **While** command to see whether the condition following it is still true. If it is, the program again executes the commands between **While** and **End** (commands 1) and then returns to the **While** command to see whether the condition following it is still true. If the condition following the **While** command is false, the program skips the commands between **While** and **End** (commands 1) and then moves on to the next command in the program (command 2).

To make the **While** command work, the commands appearing between **While** and **End** (commands 1) must change the value of the variable used in the condition that follows the **While** command. If the value of this variable does not change and the condition is true, you wind up in an *infinite loop*. That is, the calculator continues to execute the **While** command until you stop it or the batteries die.



If you find that your program inadvertently contains an infinite loop (or if it is just taking too long to execute the program and you'd like to stop the execution), press **[ON]**. You are then confronted with the ERR: BREAK error message, which gives you the option to **Quit** the execution of the program.

An example of using the **While . . . End** command appears in the second picture in Figure B6-5. The first two lines of this program don't appear in this picture. They are the same as the first two lines in the program in the second picture in Figure B6-4. The program in this picture starts with the given integer N and divides it by 2 if it is even; if it isn't, it adds 3 to the N. The program then takes the resulting number and divides it by 2 if it is even, or adds 3 to it if it isn't. This process continues until the resulting number is 1. The first **End** command appearing in this program marks the end of the **If . . . Then . . . Else . . . End** command; the second marks the end of the **While . . . End** command.

<pre>PROGRAM:WHILE :While condition :commands 1 :End :command 2</pre>	<pre>PROGRAM:DIWVHILE :While N>1 :If fPart(N/2)=0 :Then:N/2+N :Else:N+3+N :End :Disp N :End</pre>	<pre>PROGRAM:DIVREPT :Repeat N&1 :If fPart(N/2)=0 :Then:N/2+N :Else:N+3+N :End :Disp N :End</pre>
Structure	Example	Example

Figure B6-5: Using the **While . . . End** and **Repeat . . . End** commands.

The Repeat . . . End Command

The **While . . . End** and **Repeat . . . End** commands are similar, but opposite. They are similar because they have the same structure (refer to Figure B6-5). And they are opposite because the **While . . . End** command executes a block of commands *while* the specified condition is true, whereas the **Repeat . . . End** command executes a block of commands *until* the specified condition is true.

In a **Repeat . . . End** command, if the condition following the **Repeat** command is false, the program executes the commands between **Repeat** and **End** and then returns to the **Repeat** command to see whether the condition following it is still false. If it is, the program will again execute the commands between **Repeat** and **End** and then return to the **Repeat** command to see whether the condition following it is still false. If the condition following the **Repeat** command is true, the program skips the commands between **Repeat** and **End** and then moves on to the next command in the program.

Refer to the third picture in Figure B6-5 for an example of using the **Repeat . . . End** command. (The program in this picture is the same one described at the end of the previous subsection.)

The For . . . End Command

The structure of the **For . . . End** command appears in the first picture in Figure B6-6. When the **For** command is first encountered by your program, it assigns the variable **var** the value in **start** and then executes the commands appearing between **For** and **End** (commands 1). It then adds the increment **inc** to the variable **var**. If **var** is less than or equal to the value in **stop**, the process is repeated. If it isn't, the program moves on with the program by executing the command appearing after **End** (command 2).

<pre>PROGRAM:FOR :For(var,start,s toP,inc) :commands 1 :End :command 2</pre>	<pre>PROGRAM:FORPRGM :I→A:2→B :For(I,0,6,2) :I+A→A :I+B→B :Disp A+B :End</pre>	<pre>PrgmFORPRGM 15 27 Done</pre>
Structure	Example	Result

Figure B6-6: Using the For . . . End command.

An example of using the **For . . . End** command appears in the second picture of Figure B6-6. The results of executing this program appear in the third picture in this figure.

Using Branching Commands

The calculator can handle two branching commands: **Goto** and **Menu**. This section describes how to use them in a program.

Using the Goto command

The **Goto** command is used in conjunction with the **Lbl** (label) command. The **Goto** command sends the program to the corresponding **Lbl** command. The program then executes the commands that follow the **Lbl** command. To ensure that the program knows which label (**Lbl**) to go to, be sure to give the label a one- or two-character name that consists of letters, numbers, or the Greek letter θ . The **Goto** command then refers to this name when telling the program which label (**Lbl**) to go to, as shown in Figures B6-7 and B6-8. The **Goto** command directs the program to a subroutine contained in the program, or terminates the program when a specified condition is satisfied. These situations are explained in the remainder of this subsection.

The structure for using the **Goto** command to direct the program to a subroutine contained in the program appears in the first picture in Figure B6-7. The subroutine consists of the commands that are designated by commands 2 in this picture. The program in this picture executes commands 1, executes commands 2, and then (if the condition following the **If** command is true), it executes commands 2 again. It continues to re-execute commands 2 until the condition following the **If** command is false. Then it continues with the program by executing commands 3.

An example of using the **Goto** command to execute a subroutine appears in the second picture in Figure B6-7. At the beginning of the program, the user of the program is asked to enter an integer. The program then checks to make sure an integer was entered. If an integer was not entered, the program displays the message "Enter Integer," and then returns the user to the beginning of the program, once again asking the user to enter an integer. If an integer is entered, the program continues with the commands that

come after the **If . . . Then . . . End** command appearing in this picture. The request to have the user enter an integer constitutes the subroutine in this program.



When the **Goto** command directs a program to a label (**Lbl**), that label can appear in the program either before or after the **Goto** command. If it appears after the **Goto** command, the program skips executing all commands that are between the **Goto** command and the corresponding **Lbl** command.

<pre>PROGRAM:GOTO :commands 1 :Lbl 0 :commands 2 :If condition :Goto 0 :commands 3</pre>	<pre>PROGRAM:GOTOSUB :Lbl 0 :Input "INT=",N :If fPart(N)≠0 :Then :Disp "ENTER INT :EGER" :Goto 0:End</pre>
--	--

Structure

Example

Figure B6-7: Using the Goto command to execute a subroutine.

The structure for using the **Goto** command to terminate a program appears in the first picture in Figure B6-8. In this theoretical program, the program executes commands 1, and then it continually executes commands 2 until the condition after the **If** command is false. The program is terminated by the **Stop** command only when the condition appearing after the **If** command is false.

An example of a program that uses the **Goto** command to terminate a program appears in the second picture in Figure B6-8. The program in this picture asks the user to enter a number. If the number is less than 1,000, the program displays the square of that number and then prompts the user for another number. The program continues in this fashion until the user enters a number that is greater than or equal to 1,000.

<pre>PROGRAM:GOTO2 :commands 1 :Lbl 0 :commands 2 :If condition :Stop :Goto 0</pre>	<pre>PROGRAM:GOTOSTOP :Lbl 0 :Input A :If A<1000 :Stop :Disp A² :Goto 0</pre>
---	--

Structure

Example

Figure B6-8: Using the Goto command to terminate a program.

Creating a menu

The **Menu** command is a glorified **Goto** command. It allows the program user to select an item from a menu, and then have the program execute the commands that are specific to that item. After executing the commands that are specific to the chosen item, the program can terminate, return to the menu so the user can make another selection, or it can continue by executing the commands in the program that appear after the commands that are specific to the chosen menu item.

The first two pictures in Figure B6-9 illustrate the structure of a menu-driven program that terminates after executing the commands associated with the chosen menu item. If, for example, the user of this theoretical program selects itemA from the menu, commands 1 are executed, and then the **Stop** command terminates the program. If the user selects QUIT from the menu, the program clears the Home screen and then terminates because it has no more commands to execute.

The third picture in Figure B6-9 illustrates the menu that the user of the program sees. The moving broken line in the upper-right corner is the calculator's way of telling the user that it is waiting for a menu item to be selected.



When you create a menu-driven program, it's a common courtesy to offer QUIT as a menu item. This allows the user to quickly exit the program if he or she inadvertently selects the wrong program to execute.

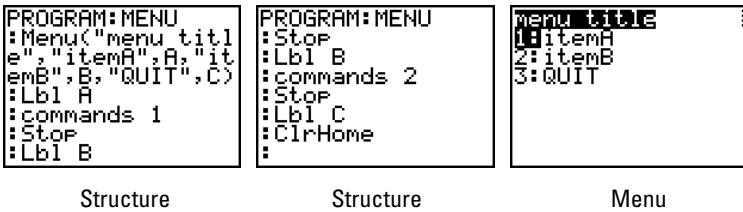


Figure B6-9: A terminating menu-driven program.

The first two pictures in Figure B6-10 illustrate the structure of a menu-driven program that returns the user to the menu after they have selected and executed a menu item. If, for example, the user of this program selects THIS from the menu, the calculator executes

the commands housed in the external program named THIS, and then returns the user to the menu to make another selection. The external program named THIS is pictured, in its entirety, in the third picture in Figure B6-10. If the user selects QUIT from the menu, the program will clear the Home screen and terminate because there are no more commands in the program for it to execute.



When you create a menu-driven program that repeatedly returns the program user to the menu, it's wise to supply the program with a means of terminating itself. Adding a QUIT option to the menu is an easy way to do so.

<pre>PROGRAM:THISTHAT :Lbl 0 :Menu("THIS THAT ", "THIS",A, "THAT ",B, "QUIT",C) :Lbl A :PrmTHIS :Goto 0</pre>	<pre>PROGRAM:THISTHAT :Goto 0 :Lbl B :PrmTHAT :Goto 0 :Lbl C :ClrHome :</pre>	<pre>PROGRAM:THIS :ClrHome :Disp "THIS" :Output(7,3, "PRE SS ENTER") :Output(8,3, "TO CONTINUE") :Pause</pre>
Structure	Structure	Called program

Figure B6-10: A menu-driven program that returns the user to the menu.

Stopping a Program

To stop a program while it is executing, press **ON**. You are then confronted with the ERR: BREAK error message that gives you the option to Quit the execution of the program.

The control command **Stop** is added to a program when you want to terminate the program before it reaches the end. It is illustrated in Figure B6-8 and in the first two pictures of Figure B6-9.

Placing the **Stop** command at the end of a program isn't necessary. The program automatically terminates execution when it reaches the last command.

Pausing a Program

When a program is executed, the output from the program is displayed very quickly on the Home screen or in a graphing window. Sometimes it is necessary to pause the program so that the program user has time to view the results of a program output.

The **Pause** command temporarily suspends the execution of a program so that the user can see the program output. The execution of the program is resumed when the program user presses `[ENTER]`, as in the program in the third picture in Figure B6-10. The program output appears in Figure B6-11. The moving broken line in the upper-right corner of Figure B6-11 tells the program user that the program is waiting for the user to press `[ENTER]` to resume execution of the program.

Because most program users don't realize that they must press `[ENTER]` to resume the execution of a paused program, I like to precede the **Pause** command in the program with the reminder that the user must "press enter to continue," as illustrated in the third picture in Figure B6-10. The consequence of doing this appears in Figure B6-11.

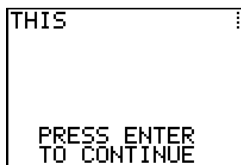


Figure B6-11: A paused program.

Executing an External Program as a Subroutine

It's quite easy to have a program call and execute another program saved on your calculator, and then return to the original program to complete its execution of that program. One command accomplishes the processes of calling, executing, and returning: the **prgm** command (accessed by pressing `[PRGM][ALPHA][x-1]`). The name of the program being called is placed directly after the command, as in the first two pictures in Figure B6-10. Notice that there is no space between the command **prgm** and the name of the program.

After the externally called program is executed, the calling program continues to execute the commands that follow the **prgm** command *provided that* the externally called program does not encounter the **Stop** command. This command terminates both the called and

calling programs. As an example, if the program `GOTOSTOP` in the second picture in Figure B6-8 is called by your program, then when the program user enters a number greater than or equal to 1000, both the calling and called programs terminate.

If you want the externally called program to return control to the calling program *before* it completes its execution, you do so by putting the **Return** command in the appropriate place in the externally called program. As an example, consider the program `GOTORTRN` appearing in Figure B6-12. `GOTORTRN` is simply the program `GOTOSTOP` (second picture in Figure B6-8) with the **Stop** command replaced by the **Return** command. If your program calls `GOTORTRN`, then when the program user enters a number greater than or equal to 1000, the `GOTORTRN` program is terminated and the calling program continues to execute.

```
PROGRAM:GOTORTRN
:Lbl 0
:Input A
:If A≥1000
:Return
:Disp A
:Goto 0
:
```

Figure B6-12: Using the Return command in a called program.



If a program containing a **Stop** command is called by another program, that command may terminate the execution of *both* programs. If the **Stop** commands in the called program are replaced with the **Return** command, then after the called program is executed, program control returns to the calling program.