

Chapter 10

More Input and Output

Some programs run fine with no input from the keyboard (the custom mode setter program, for example) but the vast majority require information from the user.

There are quite a few tools available to help with the process of effectively and concisely requesting input from the user, a process that requires skill on the part of the programmer. In this chapter, you will examine some of these tools and techniques.

CILCD

It is good to clear everything from the home screen before displaying anything so the user does not have any distractions left over from earlier work. Sometimes you might clear the screen repeatedly during execution; sometimes it is better to let the previous screen contents scroll (roll or disappear) off the top edge. After execution of this statement, nothing will be on the home screen. There are no options with **CILCD**.

Input

Once the screen is clear, you are ready to ask the user for input. You have already used **Disp** to do this, but, since **Disp** and **Input** are so often used together, Texas Instruments has given the **Input** statement syntax with an option:

```
Input ["message",]variable
```

The rectangular brackets indicate that you may include the quoted message of your choice, followed by a comma. Whether you put a message or not, a variable is required. Here are some examples of legal **Input** statements:

```
Input G  
Input "Enter x-coord:",X
```

The first **Input** statement causes only a question mark (?) to be displayed, so it should be preceded by a **Disp** statement giving directions. Otherwise, how will the user know what to type? The second **Input** statement is typical of conversational programming, which causes the screen to resemble a dialog between the user and the program.

The two statements cause two values (typed by the user) to be stored into variables **G** and **X** (decided upon by the programmer). The values that go into **G** and **X** depend on what the user types. To see the values in **G** and **X**, use **Disp**.

InpSt

In addition to numeric data, the TI-86 also allows you to enter *string* data (names and the like). This is done via the **InpSt** statement (short for INPut STring), whose general form is identical to that of **InpUt**. The following line of code contains the more complex version of the syntax that can be used with the **InpSt** statement. This statement can be used to personalize your program, but its power is considerably greater when coupled with the built-in string functions.

```
InpSt"Enter your name:",NAME
```

Note: The comma separates the message from the variable.


Because this statement requires string input, it will not accept quotation marks around the input.

Disp

Disp provides a view into the memory of the TI-86. Whatever the user types in response to a dialog, the value may eventually disappear from the screen, but it will stay in its memory location until changed (or deleted). You could see all three of the variables in the previous code examples using the following statement:

```
Disp "Your values are:",G,X,NAME
```

You might like to see all three variable values on the same line of output, but **Disp** does not work like that. The commas that separate consecutive items to be displayed cause each item to be printed on its own line on the home screen. If the **Disp** statement above were executed, the screen would look like Figure 10.1.



```
Your values are:
LEO
98
3.2
```

Figure 10.1
Output from Disp Statement

Not a pretty sight. The output in Figure 10.1 can look better, almost customized, if you use the **Outpt** statement, which is discussed on the next page.

As suggested in the sample **Disp** statement above, the general form of the **Disp** statement is:

```
Disp item[,item...]
```

Item may be variable or constant numeric or string data. It may even be one of the numeric data aggregates (list, vector, array), but **Pause** works better with them. Please note that the brackets indicate that you may put a comma and another item. The ellipsis (three dots: ...) indicates that you may continue adding commas and variables repeatedly, but the TI-86's screen is limited to seven lines of output.

Prompt

You will like the **Prompt** statement, whose simple syntax belies its power:

```
Prompt numeric variable[,numeric variable...]
```

Prompt works almost exactly like **Input** but it prints the name of the variable prior to requesting input. This can eliminate the need for a separate prompt, if the variable names used are “self-identifying.” In the program **Pyth**, you could have used the following statement:

```
Prompt Leg1,Leg2
```

You can decide if a **Disp** statement before this statement is needed. When executed alone, the preceding **Prompt** statement will cause the screen to look like Figure 10.2.

In this example, the user has entered 3 and 4 in response to the “questions” about the legs.

Since you are limited to eight-character variable names, the **Prompt** statement is certainly no replacement for the **Disp** or **Input** statements in asking for input, but it is an interesting tool.



Figure 10.2
The Output and Input from Prompt

Output

The **Outpt** statement gives you great flexibility with respect to output. The fairly simple syntax is:

```
Outpt(line,column,Item)
```

Since the screen has 8 lines and 21 columns, *line* and *column* must be integers in the intervals [1,8] and [1,21], respectively. *Item* may be any numeric or string variable, expression, or constant. Lists, vectors, and matrices are also legal *items*, but might be hard to display properly. (Consider **Pause** for this). It is impractical to use **Disp** and the variations of **Input** along with **Output** since the TI-86 makes successive **Disp** and **Input** occur on consecutive lines irrespective of where any **Output** may have occurred. You have no direct control over this. The **Output** statement puts its output where you say and returns the cursor to the line following the previous command, as shown in Figure 10.3.

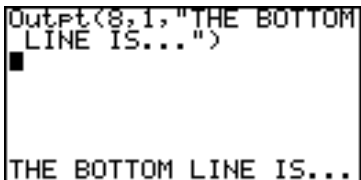


Figure 10.3
Output Statement

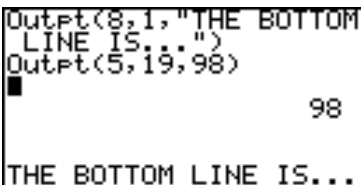


Figure 10.4
Output Statement Allows Random Cursor Control

To see how **Outpt** works, go to the home screen, press `[CLEAR]`, and type the text shown in Figure 10.4. The **Outpt** statement allows direct or random cursor control—note that the 98 is not in the last column of its line.

Summary

That is the whole input/output story in terms of the home screen. Do not forget **Menu**, which is a sort of hybrid of input, output, and control. The fun (*challenge*) really begins when you try to put output on the graph screen and get input from the user there, too.