Because `maritalStatus` is not `"s"`, we move to the `else` branch of the outer `if` statement (line 25).

```
19 if maritalStatus == "s" :
20    if income <= RATE1_SINGLE_LIMIT :
21       tax1 = RATE1 * income
22    else :
23       tax1 = RATE1 * RATE1_SINGLE_LIMIT
24       tax2 = RATE2 * (income - RATE1_SINGLE_LIMIT)
25 else :
```

Because `income` is not `<= 64000`, we move to the `else` branch of the inner `if` statement (line 28).

```
26    if income <= RATE1_MARRIED_LIMIT :
27       tax1 = RATE1 * income
28    else :
29       tax1 = RATE1 * RATE1_MARRIED_LIMIT
30       tax2 = RATE2 * (income - RATE1_MARRIED_LIMIT)
```

The values of `tax1` and `tax2` are updated.

```
28    else :
29       tax1 = RATE1 * RATE1_MARRIED_LIMIT
30       tax2 = RATE2 * (income - RATE1_MARRIED_LIMIT)
```

tax1	tax2	income	marital status
~~0~~	~~0~~	80000	m
6400	4000		

The sum `totalTax` is computed and printed. Then the program ends.

```
32 totalTax = tax1 + tax2
   . . .
35 print("The tax is $%.2f" % totalTax)
```

tax1	tax2	income	marital status	total tax
~~0~~	~~0~~	80000	m	
6400	4000			10400

Because the program trace shows the expected output ($10,400), it successfully demonstrated that this test case works correctly.

3.4 Multiple Alternatives

Multiple `if` statements can be combined to evaluate complex decisions.

In Section 3.1, you saw how to program a two-way branch with an `if` statement. In many situations, there are more than two cases. In this section, you will see how to implement a decision with multiple alternatives.

For example, consider a program that displays the effect of an earthquake, as measured by the Richter scale (see Table 4).

The 1989 Loma Prieta earthquake that damaged the Bay Bridge in San Francisco and destroyed many buildings measured 7.1 on the Richter scale.

Table 4 Richter Scale

Value	Effect
8	Most structures fall
7	Many buildings destroyed
6	Many buildings considerably damaged, some collapse
4.5	Damage to poorly constructed buildings

The Richter scale is a measurement of the strength of an earthquake. Every step in the scale, for example from 6.0 to 7.0, signifies a tenfold increase in the strength of the quake.

In this case, there are five branches: one each for the four descriptions of damage, and one for no destruction. Figure 4 shows the flowchart for this multiple-branch statement.

You could use multiple `if` statements to implement multiple alternatives, like this:

```
if richter >= 8.0 :
   print("Most structures fall")
else :
   if richter >= 7.0 :
      print("Many buildings destroyed")
   else :
      if richter >= 6.0 :
         print("Many buildings considerably damaged, some collapse")
      else :
         if richter >= 4.5 :
            print("Damage to poorly constructed buildings")
         else :
            print("No destruction of buildings")
```

but this becomes difficult to read and, as the number of branches increases, the code begins to shift further and further to the right due to the required indentation. Python provides the special construct `elif` for creating `if` statements containing multiple branches. Using the `elif` statement, the above code segment can be rewritten as

```
if richter >= 8.0 :
   print("Most structures fall")
elif richter >= 7.0 :
   print("Many buildings destroyed")
elif richter >= 6.0 :
   print("Many buildings considerably damaged, some collapse")
elif richter >= 4.5 :
   print("Damage to poorly constructed buildings")
else :
   print("No destruction of buildings")
```

As soon as one of the four tests succeeds, the effect is displayed, and no further tests are attempted. If none of the four cases applies, the final `else` clause applies, and a default message is printed.

Here you must sort the conditions and test against the largest cutoff first.

Suppose we reverse the order of tests:

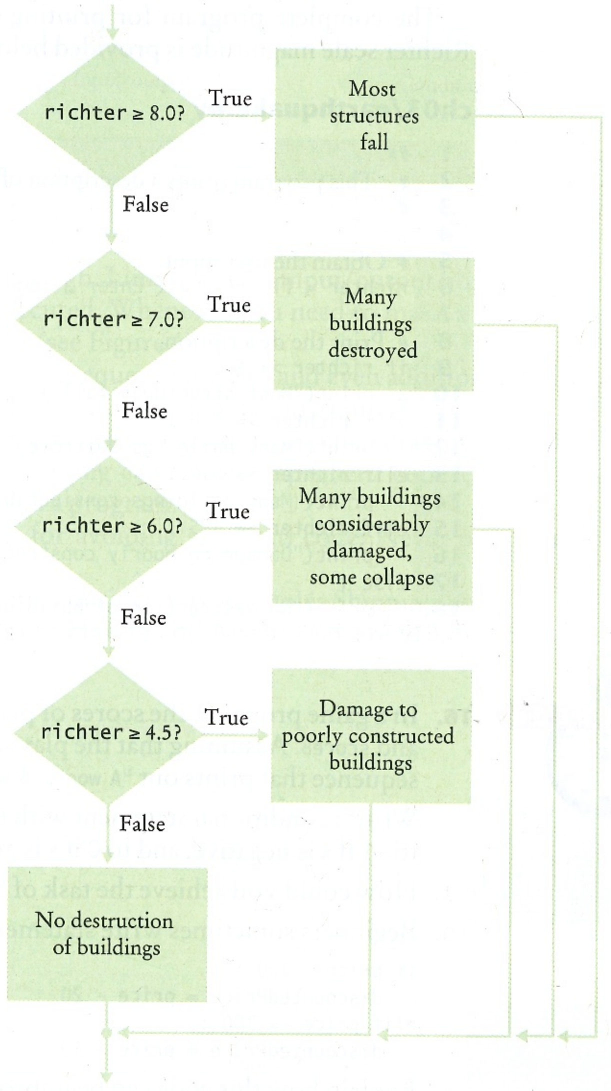
```
if richter >= 4.5 :    # Tests in wrong order
   print("Damage to poorly constructed buildings")
elif richter >= 6.0 :
   print("Many buildings considerably damaged, some collapse")
elif richter >= 7.0 :
   print("Many buildings destroyed")
elif richter >= 8.0 :
   print("Most structures fall")
```

When using multiple `if` statements, test general conditions after more specific conditions.

This does not work. Suppose the value of `richter` is 7.1. That value is at least 4.5, matching the first case. The other tests will never be attempted.

The remedy is to test the more specific conditions first. Here, the condition `richter >= 8.0` is more specific than the condition `richter >= 7.0`, and the condition `richter >= 4.5` is more general (that is, fulfilled by more values) than either of the first two.

Figure 4
Multiple Alternatives



In this example, it is also important that we use an `if/elif` sequence, not just multiple independent `if` statements. Consider this sequence of independent tests.

```
if (richter >= 8.0) :   # Didn't use else
   print("Most structures fall")
if richter >= 7.0 :
   print("Many buildings destroyed")
if richter >= 6.0 :
   print("Many buildings considerably damaged, some collapse")
if richter >= 4.5 :
   print("Damage to poorly constructed buildings")
```

Now the alternatives are no longer exclusive. If `richter` is 7.1, then the last *three* tests all match, and three messages are printed.

The complete program for printing the description of an earthquake given the Richter scale magnitude is provided below.

ch03/earthquake.py

```
##
#  This program prints a description of an earthquake, given the Richter scale magnitude.
#

# Obtain the user input.
richter = float(input("Enter a magnitude on the Richter scale: "))

# Print the description.
if richter >= 8.0 :
   print("Most structures fall")
elif richter >= 7.0 :
   print("Many buildings destroyed")
elif richter >= 6.0 :
   print("Many buildings considerably damaged, some collapse")
elif richter >= 4.5 :
   print("Damage to poorly constructed buildings")
else :
   print("No destruction of buildings")
```

SELF CHECK

16. In a game program, the scores of players A and B are stored in variables `scoreA` and `scoreB`. Assuming that the player with the larger score wins, write an `if/elif` sequence that prints out `"A won"`, `"B won"`, or `"Game tied"`.
17. Write a conditional statement with three branches that sets `s` to 1 if `x` is positive, to –1 if `x` is negative, and to `0` if `x` is zero.
18. How could you achieve the task of Self Check 17 with only two branches?
19. Beginners sometimes write statements such as the following:

```
if price > 100 :
   discountedPrice = price - 20
elif price <= 100 :
   discountedPrice = price - 10
```

    Explain how this code can be improved.
20. Suppose the user enters `-1` into the earthquake program. What is printed?
21. Suppose we want to have the earthquake program check whether the user entered a negative number. What branch would you add to the `if` statement, and where?

Practice It Now you can try these exercises at the end of the chapter: R3.22, P3.9, P3.34.

3.5 Problem Solving: Flowcharts

Flow charts are made up of elements for tasks, input/output, and decisions.

You have seen examples of flowcharts earlier in this chapter. A flowchart shows the structure of decisions and tasks that are required to solve a problem. When you have to solve a complex problem, it is a good idea to draw a flowchart to visualize the flow of control. The basic flowchart elements are shown in Figure 5.