c)
```
switch (n) {
   case  :
      cout <<                    << endl;
   case  :
      cout <<                    << endl;
      break;
   default:
      cout <<                              << endl;
}
```

d) The following code should print the values 1 to 10.
```
unsigned int n{ };
while (n <    ) {
   cout << n++ << endl;
}
```

Answers to Self-Review Exercises

5.1 a) `for`, `while`. b) after. c) `switch`. d) `continue`. e) `&&` (conditional AND). f) `false`.

5.2 a) False. The `default` case is optional. Nevertheless, it's considered good software engineering to always provide a `default` case.

b) False. The `break` statement is used to exit the `switch` statement. The `break` statement is not required when the `default` case is the last case. Nor will the `break` statement be required if having control proceed with the next case makes sense.

c) False. When using the `&&` operator, both of the relational expressions must be `true` for the entire expression to be `true`.

d) True.

5.3 a)
```
unsigned int sum{ };
for (unsigned int count{ }; count <=   ; count +=   ) {
   sum += count;
}
```

b)
```
cout << fixed << left
   << setprecision( ) << setw(  ) <<
   << setprecision( ) << setw(  ) <<
   << setprecision( ) << setw(  ) <<          << endl;
```
Output is:
```
333.5          333.55          333.546
```

c)
```
cout << fixed << setprecision( ) << setw(  ) << pow(   ,   ) << endl;
```
Output is:
```
    15.63
```

d)
```
unsigned int x{ };
while (x <=    ) {
   if (x %    ==  ) {
      cout << x << endl;
   }
   else {
      cout << x <<      ;
   }

   ++x;
}
```

e)
```
for (unsigned int x = 1; x <= 20; ++x) {
   if (x % 5 == 0) {
      cout << x << endl;
   }
   else {
      cout << x << '\t';
   }
}
```

5.4 a) *Error:* The semicolon after the `while` header causes an infinite loop.
Correction: Delete the semicolon after the `while` header.

b) *Error:* Using a floating-point number to control a `for` iteration statement.
Correction: Use an `unsigned int` and perform the proper calculation to get the values.

```
for (unsigned int y = 1; y != 10; ++y) {
   cout << (static_cast< double >(y) / 10) << endl;
}
```

c) *Error:* Missing `break` statement in the first case.
Correction: Add a `break` statement at the end of the first case. This is not an error if you want the statement of `case 2:` to execute every time the `case 1:` statement executes.

d) *Error:* Improper relational operator used in the loop-continuation condition.
Correction: Use `<=` rather than `<`, or change `10` to `11`.

Exercises

5.5 Describe the four basic elements of counter-controlled iteration.

5.6 Compare and contrast the `while` and `for` iteration statements.

5.7 Discuss a situation in which it would be more appropriate to use a `do…while` statement than a `while` statement. Explain why.

5.8 Compare and contrast the `break` and `continue` statements.

5.9 *(Find the Code Errors)* Find the error(s), if any, in each of the following:

a)
```
For (unsigned int x{100}, x >= 1, ++x) {
   cout << x << endl;
}
```

b) The following code should print whether integer `value` is odd or even:

```
switch (value % 2) {
   case 0:
      cout << "Even integer" << endl;
   case 1:
      cout << "Odd integer" << endl;
}
```

c) The following code should output the odd integers from 19 to 1:

```
for (unsigned int x{19}; x >= 1; x += 2) {
   cout << x << endl;
}
```

d) The following code should output the even integers from 2 to 100:

```
unsigned int counter{2};

do {
   cout << counter << endl;
   counter += 2;
} While ( counter < 100 );
```

5.10 What does the following program do?

```
// Exercise 5.10: Printing.cpp
#include <iostream>
using namespace std;

int main() {
   for (int i{1}; i <= 10; i++) {
      for (int j{1}; j <= 5; j++) {
         cout << '@';
      }

      cout << endl;
   }
}
```

5.11 *(Find the Smallest Value)* Write an application that finds the smallest of several integers. Assume that the first value read specifies the number of values to input from the user.

5.12 *(Calculating the Product of Odd Integers)* Write an application that calculates the product of the odd integers from 1 to 15.

5.13 *(Factorials) Factorials* are used frequently in probability problems. The factorial of a positive integer *n* (written *n*! and pronounced "*n* factorial") is equal to the product of the positive integers from 1 to *n*. Write an application that calculates the factorials of 1 through 20. Use type `long`. Display the results in tabular format. What difficulty might prevent you from calculating the factorial of 100?

5.14 *(Modified Compound-Interest Program)* Modify the compound-interest application of Fig. 5.6 to repeat its steps for interest rates of 5%, 6%, 7%, 8%, 9% and 10%. Use a `for` loop to vary the interest rate.

5.15 *(Triangle-Printing Program)* Write an application that displays the following patterns separately, one below the other. Use `for` loops to generate the patterns. All asterisks (`*`) should be printed by a single statement of the form `cout << '*';` which causes the asterisks to print side by side. A statement of the form `cout << '\n';` can be used to move to the next line. A statement of the form `cout << ' ';` can be used to display a space for the last two patterns. There should be no other output statements in the program. [*Hint:* The last two patterns require that each line begin with an appropriate number of blank spaces.]

```
(a)          (b)          (c)          (d)
*            **********   **********            *
**           *********     *********           **
***          ********       ********          ***
****         *******         *******         ****
*****        ******           ******        *****
******       *****             *****       ******
*******      ****               ****      *******
********     ***                 ***     ********
*********    **                   **    *********
**********   *                     *   **********
```

5.16 *(Bar-Chart Printing Program)* One interesting application of computers is to display graphs and bar charts. Write an application that reads five numbers between 1 and 30. For each number that's read, your program should display the same number of adjacent asterisks. For example, if your program reads the number 7, it should display *******. Display the bars of asterisks *after* you read all five numbers.

5.17 *(Calculating Sales)* An online retailer sells five products whose retail prices are as follows: Product 1, $2.98; product 2, $4.50; product 3, $9.98; product 4, $4.49 and product 5, $6.87. Write an application that reads a series of pairs of numbers as follows:

a) product number
b) quantity sold

Your program should use a `switch` statement to determine the retail price for each product. It should calculate and display the total retail value of all products sold. Use a sentinel-controlled loop to determine when the program should stop looping and display the final results.

5.18 Assume that `i = 1`, `j = 2`, `k = 3` and `m = 2`. What does each of the following statements print?

a) `cout << (i == 1) << endl;`
b) `cout << (j == 3) << endl;`
c) `cout << (i >= 1 && j < 4) << endl;`
d) `cout << (m <= 99 && k < m) << endl;`
e) `cout << (j >= i || k == m) << endl;`
f) `cout << (k + m < j || 3 - j >= k) << endl;`
g) `cout << (!m ) << endl;`
h) `cout << (!( j - m)) << endl;`
i) `cout << (!(k > m)) << endl;`

5.19 *(Calculating the Value of π)* Calculate the value of π from the infinite series

$$\pi = 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \frac{4}{11} + \cdots$$

Print a table that shows the value of π approximated by computing the first 200,000 terms of this series. How many terms do you have to use before you first get a value that begins with 3.14159?

5.20 *(Pythagorean Triples)* A right triangle can have sides whose lengths are all integers. The set of three integer values for the lengths of the sides of a right triangle is called a Pythagorean triple. The lengths of the three sides must satisfy the relationship that the sum of the squares of two of the sides is equal to the square of the hypotenuse. Write an application that displays a table of the Pythagorean triples for `side1`, `side2` and the `hypotenuse`, all no larger than 500. Use a triple-nested `for` loop that tries all possibilities. This is an example of "brute-force" computing. You'll learn in more advanced computer-science courses that for many interesting problems there's no known algorithmic approach other than using sheer brute force.

5.21 *(Modified Triangle-Printing Program)* Modify Exercise 5.15 to combine your code from the four separate triangles of asterisks such that all four patterns print side by side. [*Hint:* Make clever use of nested `for` loops.]

5.22 *(De Morgan's Laws)* In this chapter, we discussed the logical operators `&&`, `||` and `!`. De Morgan's laws can sometimes make it more convenient for us to express a logical expression. These laws state that the expression `!(`*condition1* `&&` *condition2*`)` is logically equivalent to the expression `(!`*condition1* `||` `!`*condition2*`)`. Also, the expression `!(`*condition1* `||` *condition2*`)` is logically equivalent to the expression `(!`*condition1* `&&` `!`*condition2*`)`. Use De Morgan's laws to write equivalent expressions for each of the following, then write an application to show that both the original expression and the new expression in each case produce the same value:

a) `!(x < 5) && !(y >= 7)`
b) `!(a == b) || !(g != 5)`
c) `!((x <= 8) && (y > 4))`
d) `!((i > 4) || (j <= 6))`

5.23 ***(Diamond-Printing Program)*** Write an application that prints the following diamond shape. You may use output statements that print a single asterisk (*), a single space or a single newline character. Maximize your use of iteration (with nested `for` statements), and minimize the number of output statements.

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

5.24 ***(Modified Diamond-Printing Program)*** Modify the application you wrote in Exercise 5.23 to read an odd number in the range 1 to 19 to specify the number of rows in the diamond. Your program should then display a diamond of the appropriate size.

5.25 ***(Removing `break` and `continue`)*** A criticism of the `break` statement and the `continue` statement is that each is *unstructured*. Actually, these statements can *always* be replaced by structured statements, although doing so can be awkward. Describe in general how you'd remove any `break` statement from a loop in a program and replace it with some structured equivalent. [*Hint:* The `break` statement exits a loop from the body of the loop. The other way to exit is by failing the loop-continuation test. Consider using in the loop-continuation test a second test that indicates "early exit because of a 'break' condition."] Use the technique you develop here to remove the `break` statement from the application in Fig. 5.13.

5.26 What does the following program segment do?

```
for (unsigned int i{ }; i <=  ; i++) {
   for (unsigned int j{ }; j <=  ; j++) {
      for (unsigned int k{ }; k <=  ; k++) {
         cout <<    ;
      }

      cout << endl;
   }

   cout << endl;
}
```

5.27 ***(Replacing `continue` with a Structured Equivalent)*** Describe in general how you'd remove any `continue` statement from a loop in a program and replace it with some structured equivalent. Use the technique you develop here to remove the `continue` statement from the program in Fig. 5.14.

5.28 ***("The Twelve Days of Christmas" Song)*** Write an application that uses iteration and `switch` statements to print the song "The Twelve Days of Christmas." One `switch` statement should be used to print the day ("first," "second," and so on). A separate `switch` statement should be used to print the remainder of each verse. Visit the website `en.wikipedia.org/wiki/The_Twelve_Days_of_Christmas_(song)` for the lyrics of the song.

5.29 ***(Peter Minuit Problem)*** Legend has it that, in 1626, Peter Minuit purchased Manhattan Island for $24.00 in barter. Did he make a good investment? To answer this question, modify the compound-interest program of Fig. 5.6 to begin with a principal of $24.00 and to calculate the amount of interest on deposit if that money had been kept on deposit until this year (e.g., 390 years through 2016). Place the `for` loop that performs the compound-interest calculation in an outer `for` loop that varies the interest rate from 5% to 10% to observe the wonders of compound interest.

5.30 *(`DollarAmount` Constructor with Two Parameters)* Enhance class `DollarAmount` (Fig. 5.8) with a constructor that receives two parameters representing the whole number of dollars and the whole number of cents. Use these to calculate and store in the data member `amount` the total number of pennies. Test the class with your new constructor.

5.31 *(`DollarAmount` Arithmetic)* Enhance class `DollarAmount` from Exercise 5.30 with a `divide` member function that receives an `int` parameter, divides the data member `amount` by that value and stores the result in the data member. Use rounding techniques similar to the `addInterest` member function. Test your new `divide` member function.

5.32 *(`DollarAmount` with Banker's Rounding)* The `DollarAmount` class's `addInterest` member function uses the *biased* half-up rounding technique in which fractional amounts of .1, .2, .3 and .4 round down, and .5, .6, .7, .8 and .9 round up. In this technique, four values round down and five round up. Banker's rounding fixes this problem by rounding .5 to the nearest *even* integer—e.g., 0.5 rounds to 0, 1.5 and 2.5 round to 2, 3.5 and 4.5 round to 4, etc. Enhance class `DollarAmount` from Exercise 5.31 by reimplementing `addInterest` to use banker's rounding, then retest the compound-interest program.

5.33 *(`DollarAmount` with `dollars` and `cents` Data Members)* Reimplement class `DollarAmount` from Exercise 5.32 to store data members `dollars` and `cents`, rather than `amount`. Modify the body of each constructor and member function appropriately to manipulate the `dollars` and `cents` data members.

5.34 *(`Account` Class That Stores a `DollarAmount`)* Upgrade the `Account` class from Exercise 3.9 to define its `balance` data member as an object of class `DollarAmount` from Exercise 5.33. Reimplement the bodies of class Account's constructor and member functions accordingly.

5.35 *(Displaying the Interest Rate in the `DollarAmount` Example)* Enhance the `main` program in Fig. 5.7 to display the interest rate based on the two integers entered by the user. For example, if the user enters 2 and 100, display 2.0%, and if the user enters 2015 and 100000, display 2.015%.

5.36 *(Showing That `double` Values Are Approximate)* Create a program that assigns 123.02 to a `double` variable, then displays the variable's value with many digits of precision to the right of the decimal point. Which precision first shows you the representational error of storing 123.02 in a `double` variable?

Making a Difference

5.37 *(Global Warming Facts Quiz)* The controversial issue of global warming has been widely publicized by the film "An Inconvenient Truth," featuring former Vice President Al Gore. Mr. Gore and a U.N. network of scientists, the Intergovernmental Panel on Climate Change, shared the 2007 Nobel Peace Prize in recognition of "their efforts to build up and disseminate greater knowledge about man-made climate change." Research *both* sides of the global warming issue online (you might want to search for phrases like "global warming skeptics"). Create a five-question multiple-choice quiz on global warming, each question having four possible answers (numbered 1–4). Be objective and try to fairly represent both sides of the issue. Next, write an application that administers the quiz, calculates the number of correct answers (zero through five) and returns a message to the user. If the user correctly answers five questions, print "Excellent"; if four, print "Very good"; if three or fewer, print "Time to brush up on your knowledge of global warming," and include a list of some of the websites where you found your facts.