### *Section 4.11.7 Preventing Narrowing Conversions with List Initialization*

- For fundamental-type variables, list-initialization syntax prevents **narrowing conversions** (p. 136) that could result in data loss.

### *Section 4.12 Compound Assignment Operators*

- The **compound assignment operators** `+=`, `-=`, `*=`, `/=` and `%=` (p. 136) abbreviate assignment expressions.

### *Section 4.13 Increment and Decrement Operators*

- The **increment** (p. 137; `++`) and **decrement** (p. 137; `--`) operators increment or decrement a variable by 1, respectively. If the operator is prefixed to the variable, the variable is incremented or decremented by 1 first, then its new value is used in the expression in which it appears. If the operator is postfixed to the variable, the variable is first used in the expression in which it appears, then the variable's value is incremented or decremented by 1.

### *Section 4.14 Fundamental Types Are Not Portable*

- All variables must have a type.
- The fundamental types are not guaranteed to be identical from computer to computer. An `int` on one machine might be represented by 16 bits (2 bytes) of memory, on a second machine by 32 bits (4 bytes), and on another machine by 64 bits (8 bytes).

## Self-Review Exercises

**4.1** Answer each of the following questions.

a) All programs can be written in terms of three types of control statements: _____, _____ and _____.

b) The _____selection statement is used to execute one action when a condition is *true* or a different action when that condition is *false*.

c) Repeating a set of instructions a specific number of times is called _____ iteration.

d) When it isn't known in advance how many times a set of statements will be repeated, a(n)_____value can be used to terminate the iteration.

**4.2** Write four different C++ statements that each add 1 to integer variable `x`.

**4.3** Write C++ statements to accomplish each of the following:

a) In one statement, assign the sum of the current value of `x` and `y` to `z` and postincrement the value of `x`.

b) Determine whether the value of the variable `count` is greater than 10. If it is, print `"Count is greater than 10"`.

c) Predecrement the variable `x` by 1, then subtract it from the variable `total`.

d) Calculate the remainder after `q` is divided by `divisor` and assign the result to `q`. Write this statement two different ways.

**4.4** Write C++ statements to accomplish each of the following tasks.

a) Declare variable `sum` to be of type `unsigned int` and initialize it to `0`.

b) Declare variable `x` to be of type `unsigned int` and initialize it to `1`.

c) Add variable `x` to variable `sum` and assign the result to variable `sum`.

d) Print `"The sum is: "` followed by the value of variable `sum`.

**4.5** Combine the statements that you wrote in Exercise 4.4 into a program that calculates and prints the sum of the integers from 1 to 10. Use the `while` statement to loop through the calculation and increment statements. The loop should terminate when the value of `x` becomes 11.

**4.6** State the values of *each* of these `unsigned int` variables after the calculation is performed. Assume that, when each statement begins executing, all variables have the integer value 5.

a) `product *= x++;`
b) `quotient /= ++x;`

**4.7** Write single C++ statements or portions of statements that do the following:

a) Input `unsigned int` variable `x` with `cin` and `>>`.
b) Input `unsigned int` variable `y` with `cin` and `>>`.
c) Declare `unsigned int` variable `i` and initialize it to 1.
d) Declare `unsigned int` variable `power` and initialize it to 1.
e) Multiply variable `power` by `x` and assign the result to `power`.
f) Preincrement variable `i` by 1.
g) Determine whether `i` is less than or equal to `y`.
h) Output integer variable `power` with `cout` and `<<`.

**4.8** Write a C++ program that uses the statements in Exercise 4.7 to calculate `x` raised to the `y` power. The program should have a `while` iteration statement.

**4.9** Identify and correct the errors in each of the following:

a)
```
while (c <= 5) {
   product *= c;
   ++c;
```
b)
```
cin << value;
```
c)
```
if (gender == 1) {
   cout << "Woman" << endl;
else; {
   cout << "Man" << endl;
}
```

**4.10** What's wrong with the following `while` iteration statement?

```
while (z >= 0) {
   sum += z;
}
```

## Answers to Self-Review Exercises

**4.1** a) Sequence, selection and iteration. b) `if...else`. c) Counter-controlled or definite. d) Sentinel, signal, flag or dummy.

**4.2**
```
x = x + 1;
x += 1;
++x;
x++;
```

**4.3** a)
```
z = x++ + y;
```
b)
```
if (count > 10) {
   cout << "Count is greater than 10" << endl;
}
```
c)
```
total -= --x;
```
d)
```
q %= divisor;
q = q % divisor;
```

**4.4** a)
```
unsigned int sum{0};
```
b)
```
unsigned int x{1};
```

c) `sum += x;`
or
`sum = sum + x;`
d) `cout << "The sum is: " << sum << endl;`

**4.5** See the following code:

```
// Exercise 4.5: Calculate.cpp
// Calculate the sum of the integers from 1 to 10
#include <iostream>
using namespace std;

int main() {
   unsigned int sum{0};
   unsigned int x{1};

   while (x <= 10) { // while x is less than or equal to 10
      sum += x; // add x to sum
      ++x; // increment x
   }

   cout << "The sum is: " << sum << endl;
}
```

```
The sum is: 55
```

**4.6** a) `product = 2, x = 6;`
b) `quotient = 0, x = 6;`

**4.7** a) `cin >> x;`
b) `cin >> y;`
c) `unsigned int i{1};`
d) `unsigned int power{ };`
e) `power *= x;`
or
`power = power * x;`
f) `++i;`
g) `if (i <= y)`
h) `cout << power << endl;`

**4.8** See the following code:

```
// Exercise 4.8 Solution: power.cpp
// Raise x to the y power.
#include <iostream>
using namespace std;

int main() {
   unsigned int i{1}; // initialize i to begin counting from 1
   unsigned int power{1}; // initialize power

   cout << "Enter base as an integer: "; // prompt for base
   unsigned int x; // base
   cin >> x; // input base

```

```
   cout << "Enter exponent as an integer: "; // prompt for exponent
   unsigned int y; // exponent
   cin >> y; // input exponent

   // count from 1 to y and multiply power by x each time
   while (i <= y) {
      power *= x;
      ++i;
   } // end while

   cout << power << endl; // display result
} // end main
```

```
Enter base as an integer: 2
Enter exponent as an integer: 3
8
```

**4.9** a) *Error:* Missing the closing right brace of the `while` body.
*Correction:* Add closing right brace after the statement `++c;`.

b) *Error:* Used stream insertion instead of stream extraction.
*Correction:* Change `<<` to `>>`.

c) *Error:* Semicolon after `else` is a logic error. The second output statement always executes.
*Correction:* Remove the semicolon after `else`.

**4.10** The value of the variable z is never changed in the `while` statement. Therefore, if the loop-continuation condition (`z >= 0`) is initially *true*, an infinite loop is created. To prevent the infinite loop, z must be decremented so that it eventually becomes less than 0.

## Exercises

**4.11** *(Correct the Code Errors)* Identify and correct the error(s) in each of the following:

a)
```
if (age >= 65); {
   cout << "Age is greater than or equal to 65" << endl;
}
else {
   cout << "Age is less than 65 << endl";
}
```

b)
```
if (age >= 65) {
   cout << "Age is greater than or equal to 65" << endl;
else; {
   cout << "Age is less than 65 << endl";
}
```

c)
```
unsigned int x{1};
unsigned int total;

while (x <= 10) {
   total += x;
   ++x;
}
```

d)
```
While (x <= 100)
   total += x;
   ++x;
```

e)
```
while (y > 0) {
   cout << y << endl;
   ++y;
}
```

**4.12** *(What Does this Program Do?)* What does the following program print?

```
// Exercise 4.12: Mystery.cpp
#include <iostream>
using namespace std;

int main() {
   unsigned int x{1};
   unsigned int total{0};

   while (x <= 10) {
      int y = x * x;
      cout << y << endl;
      total += y;
      ++x;
   }

   cout << "Total is " << total << endl;
}
```

*For Exercises 4.13—4.16, perform each of these steps:*

a) Read the problem statement.
b) Formulate the algorithm using pseudocode and top-down, stepwise refinement.
c) Write a C++ program.
d) Test, debug and execute the C++ program.

**4.13** *(Gas Mileage)* Drivers are concerned with the mileage obtained by their automobiles. One driver has kept track of several trips by recording miles driven and gallons used for each trip. Develop a C++ program that uses a `while` statement to input the miles driven and gallons used for each trip. The program should calculate and display the miles per gallon obtained for each trip and print the combined miles per gallon obtained for all tankfuls up to this point.

```
Enter miles driven (-1 to quit): 287
Enter gallons used: 13
MPG this trip: 22.076923
Total MPG: 22.076923

Enter miles driven (-1 to quit): 200
Enter gallons used: 10
MPG this trip: 20.000000
Total MPG: 21.173913

Enter the miles driven (-1 to quit): 120
Enter gallons used: 5
MPG this trip: 24.000000
Total MPG: 21.678571

Enter the miles used (-1 to quit): -1
```

**4.14** *(Credit Limits)* Develop a C++ program that will determine whether a department-store customer has exceeded the credit limit on a charge account. For each customer, the following facts are available:

a) Account number (an integer)
b) Balance at the beginning of the month
c) Total of all items charged by this customer this month
d) Total of all credits applied to this customer's account this month
e) Allowed credit limit

The program should use a `while` statement to input each of these facts, calculate the new balance (= beginning balance + charges – credits) and determine whether the new balance exceeds the customer's credit limit. For those customers whose credit limit is exceeded, the program should display the customer's account number, credit limit, new balance and the message "Credit Limit Exceeded."

```
Enter account number (or -1 to quit): 100
Enter beginning balance: 5394.78
Enter total charges: 1000.00
Enter total credits: 500.00
Enter credit limit: 5500.00
New balance is 5894.78
Account:        100
Credit limit: 5500.00
Balance:      5894.78
Credit Limit Exceeded.

Enter Account Number (or -1 to quit): 200
Enter beginning balance: 1000.00
Enter total charges: 123.45
Enter total credits: 321.00
Enter credit limit: 1500.00
New balance is 802.45

Enter Account Number (or -1 to quit): -1
```

**4.15** *(Sales-Commission Calculator)* A large company pays its salespeople on a commission basis. The salespeople each receive $200 per week plus 9% of their gross sales for that week. For example, a salesperson who sells $5000 worth of chemicals in a week receives $200 plus 9% of $5000, or a total of $650. Develop a C++ program that uses a `while` statement to input each salesperson's gross sales for last week and calculates and displays that salesperson's earnings. Process one salesperson's figures at a time.

```
Enter sales in dollars (-1 to end): 5000.00
Salary is: $650.00

Enter sales in dollars (-1 to end): 6000.00
Salary is: $740.00

Enter sales in dollars (-1 to end): 7000.00
Salary is: $830.00

Enter sales in dollars (-1 to end): -1
```

**4.16** *(Salary Calculator)* Develop a C++ program that uses a `while` statement to determine the gross pay for each of several employees. The company pays "straight time" for the first 40 hours worked by each employee and pays "time-and-a-half" for all hours worked in excess of 40 hours. You are given a list of the employees of the company, the number of hours each employee worked last week and the hourly rate of each employee. Your program should input this information for each employee and should determine and display the employee's gross pay.

```
Enter hours worked (-1 to end): 39
Enter hourly rate of the employee ($00.00): 10.00
Salary is $390.00

Enter hours worked (-1 to end): 40
Enter hourly rate of the employee ($00.00): 10.00
Salary is $400.00

Enter hours worked (-1 to end): 41
Enter hourly rate of the employee ($00.00): 10.00
Salary is $415.00

Enter hours worked (-1 to end): -1
```

**4.17** *(Find the Largest)* The process of finding the largest number (i.e., the maximum of a group of numbers) is used frequently in computer applications. For example, a program that determines the winner of a sales contest inputs the number of units sold by each salesperson. The salesperson who sells the most units wins the contest. Write a C++ program that uses a `while` statement to determine and print the largest of 10 numbers input by the user. Your program should use three variables, as follows:

a) `counter`—A counter to count to 10 (i.e., to keep track of how many numbers have been input and to determine when all 10 numbers have been processed).
b) `number`—The current number input to the program.
c) `largest`—The largest number found so far.

**4.18** *(Tabular Output)* Write a C++ program that uses a `while` statement and the tab escape sequence \t to print the following table of values:

```
N       10*N    100*N   1000*N

1       10      100     1000
2       20      200     2000
3       30      300     3000
4       40      400     4000
5       50      500     5000
```

**4.19** *(Find the Two Largest Numbers)* Using an approach similar to that in Exercise 4.17, find the *two* largest values among the 10 numbers. [*Note:* You must input each number only once.]

**4.20** *(Validating User Input)* The examination-results program of Fig. 4.14 assumes that any value input by the user that's not a 1 must be a 2. Modify the application to validate its inputs. On any input, if the value entered is other than 1 or 2, keep looping until the user enters a correct value.

**4.21** *(What Does this Program Do?)* What does the following program print?

```
1  // Exercise 4.21: Mystery2.cpp
2  #include <iostream>
3  using namespace std;
4
5  int main() {
6     unsigned int count{ };
7
8     while (count <=    ) {
9        cout << (count %    ==    ?         :            ) << endl;
10       ++count;
11    }
12 }
```

**4.22** *(What Does this Program Do?)* What does the following program print?

```
// Exercise 4.22: Mystery3.cpp
#include <iostream>
using namespace std;

int main() {
   unsigned int row{10};

   while (row >= 1) {
      unsigned int column{1};

      while (column <= 10) {
         cout << (row % 2 == 1 ? "<" : ">");
         ++column;
      }

      --row;
      cout << endl;
   }
}
```

**4.23** *(Dangling-else Problem)* C++ compilers always associate an `else` with the immediately preceding `if` unless told to do otherwise by the placement of braces (`{` and `}`). This behavior can lead to what is referred to as the dangling-else problem. The indentation of the nested statement

```
if (x > 5)
   if (y > 5)
      cout << "x and y are > 5";
else
   cout << "x is <= 5";
```

appears to indicate that if `x` is greater than `5`, the nested `if` statement determines whether `y` is also greater than `5`. If so, the statement outputs the string `"x and y are > 5"`. Otherwise, it appears that if `x` is not greater than `5`, the `else` part of the `if...else` outputs the string `"x is <= 5"`. Beware! This nested `if...else` statement does *not* execute as it appears. The compiler actually interprets the statement as

```
if (x > 5)
   if (y > 5)
      cout << "x and y are > 5";
   else
      cout << "x is <= 5";
```

in which the body of the first `if` is a *nested* `if...else`. The outer `if` statement tests whether `x` is greater than `5`. If so, execution continues by testing whether `y` is also greater than `5`. If the second condition is *true*, the proper string—`"x and y are > 5"`—is displayed. However, if the second condition is *false*, the string `"x is <= 5"` is displayed, even though we know that `x` is greater than `5`. Equally bad, if the outer `if` statement's condition is *false*, the inner `if...else` is skipped and nothing is displayed. For this exercise, add braces to the preceding code snippet to force the nested `if...else` statement to execute as it was originally intended.

**4.24** *(Another Dangling-else Problem)* Based on the dangling-`else` discussion in Exercise 4.23, state the output for each of the following code snippets when `x` is `9` and `y` is `11` and when `x` is `11` and `y` is `9`. We eliminated the indentation from the following code to make the problem more challenging. [*Hint:* Apply indentation conventions you've learned.]

a)
```
if (x < 10)
if (y > 10)
cout << "*****" << endl;
else
cout << "#####" << endl;
cout << "$$$$$" << endl;
```
b)
```
if (x < 10)
{
if (y > 10)
cout << "*****" << endl;
}
else
{
cout << "#####" << endl;
cout << "$$$$$" << endl;
}
```

**4.25** *(Another Dangling-else Problem)* Based on the dangling-else discussion in Exercise 4.23, modify the following code to produce the output shown. Use proper indentation techniques. You must not make any additional changes other than inserting braces. We eliminated the indentation from the following code to make the problem more challenging. [*Note:* It's possible that no modification is necessary.]

```
if ( y == 8 )
if ( x == 5 )
cout << "@@@@@" << endl;
else
cout << "#####" << endl;
cout << "$$$$$" << endl;
cout << "&&&&&" << endl;
```

a) Assuming x = 5 and y = 8, the following output is produced.

```
@@@@@
$$$$$
&&&&&
```

b) Assuming x = 5 and y = 8, the following output is produced.

```
@@@@@
```

c) Assuming x = 5 and y = 8, the following output is produced.

```
@@@@@
&&&&&
```

d) Assuming x = 5 and y = 7, the following output is produced. [*Note:* The last three output statements after the else are all part of a block.]

```
#####
$$$$$
&&&&&
```

**4.26** *(Square of Asterisks)* Write a program that reads in the size of the side of a square, then prints a hollow square of that size out of asterisks and blanks. Your program should work for squares of all side sizes between 1 and 20. For example, if your program reads a size of 5, it should print

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

**4.27** *(Palindromes)* A palindrome is a number or a text phrase that reads the same backward as forward. For example, each of the following five-digit integers is a palindrome: 12321, 55555, 45554 and 11611. Write a program that reads in a five-digit integer and determines whether it's a palindrome. [*Hint:* Use the division and remainder operators to separate the number into its individual digits.]

**4.28** *(Printing the Decimal Equivalent of a Binary Number)* Input an integer containing only 0s and 1s (i.e., a "binary" integer) and print its decimal equivalent. Use the remainder and division operators to pick off the "binary" number's digits one at a time from right to left. Much as in the decimal number system, where the rightmost digit has a positional value of 1, the next digit left has a positional value of 10, then 100, then 1000, and so on, in the binary number system the rightmost digit has a positional value of 1, the next digit left has a positional value of 2, then 4, then 8, and so on. Thus the decimal number 234 can be interpreted as 2 * 100 + 3 * 10 + 4 * 1. The decimal equivalent of binary 1101 is 1 * 1 + 0 * 2 + 1 * 4 + 1 * 8 or 1 + 0 + 4 + 8, or 13. [*Note:* To learn more about binary numbers, refer to Appendix D.]

**4.29** *(Checkerboard Pattern of Asterisks)* Write a program that displays the following checkerboard pattern. Your program must use only three output statements, one of each of the following forms:

```
cout << "* ";
cout << ' ';
cout << endl;
```

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

**4.30** *(Multiples of 2 with an Infinite Loop)* Write a program that prints the powers of the integer 2, namely 2, 4, 8, 16, 32, 64, etc. Your `while` loop should not terminate (i.e., you should create an infinite loop). To do this, simply use the keyword *true* as the expression for the `while` statement. What happens when you run this program?

**4.31** *(Calculating a Circle's Diameter, Circumference and Area)* Write a program that reads the radius of a circle (as a `double` value) and computes and prints the diameter, the circumference and the area. Use the value 3.14159 for π.

**4.32** What's wrong with the following statement? Provide the correct statement to accomplish what the programmer was probably trying to do.

```
cout << ++( x + y );
```

**4.33** *(Sides of a Triangle)* Write a program that reads three nonzero `double` values and determines and prints whether they could represent the sides of a triangle.

4.34 *(Sides of a Right Triangle)* Write a program that reads three nonzero integers and determines and prints whether they're the sides of a right triangle.

4.35 *(Factorial)* The factorial of a nonnegative integer $n$ is written $n!$ (pronounced "$n$ factorial") and is defined as follows:

$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1$ (for values of $n$ greater than 1)

and

$n! = 1$ (for $n = 0$ or $n = 1$).

For example, $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$, which is 120. Use `while` statements in each of the following:

a) Write a program that reads a nonnegative integer and computes and prints its factorial.

b) Write a program that estimates the value of the mathematical constant $e$ by using the formula:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots$$

Prompt the user for the desired accuracy of $e$ (i.e., the number of terms in the summation).

c) Write a program that computes the value of $e^x$ by using the formula

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

Prompt the user for the desired accuracy of $e$ (i.e., the number of terms in the summation).

4.36 *(Modified `Account` Class)* Modify class `Account` (Exercise 3.9) to represent the `balance` data member as type `double`. Also, display all `double` amounts with two digits to the right of the decimal point.

## Making a Difference

4.37 *(Enforcing Privacy with Cryptography)* The explosive growth of Internet communications and data storage on Internet-connected computers has greatly increased privacy concerns. The field of cryptography is concerned with coding data to make it difficult (and hopefully—with the most advanced schemes—impossible) for unauthorized users to read. In this exercise you'll investigate a simple scheme for encrypting and decrypting data. A company that wants to send data over the Internet has asked you to write a program that will encrypt the data so that it may be transmitted more securely. All the data is transmitted as four-digit integers. Your application should read a four-digit integer entered by the user and encrypt it as follows: Replace each digit with the result of adding 7 to the digit and getting the remainder after dividing the new value by 10. Then swap the first digit with the third, and swap the second digit with the fourth. Then print the encrypted integer. Write a separate application that inputs an encrypted four-digit integer and decrypts it (by reversing the encryption scheme) to form the original number. [*Optional reading project:* Research "public key cryptography" in general and the PGP (Pretty Good Privacy) specific public-key scheme. You may also want to investigate the RSA scheme, which is widely used in industrial-strength applications.]

4.38 *(World Population Growth)* World population has grown considerably over the centuries. Continued growth could eventually challenge the limits of breathable air, drinkable water, arable cropland and other precious resources. There is evidence that growth has been slowing in recent years and that world population could peak some time this century, then start to decline.

For this exercise, research world population growth issues online. *Be sure to investigate various viewpoints.* Get estimates for the current world population and its growth rate (the percentage by which it is likely to increase this year). Write a program that calculates world population growth each year for the next 75 years, *using the simplifying assumption that the current growth rate will stay constant.* Print the results in a table. The first column should display the year from year 1 to year