Lecture 9: Control Statements (cont)

Ioan Raicu
Department of Electrical Engineering & Computer Science
Northwestern University

EECS 211
Fundamentals of Computer Programming II
April 12th, 2010
• The variable \texttt{average} is declared to be of type \texttt{double}
to capture the fractional result of our calculation.

• \texttt{total} and \texttt{gradeCounter} are both integer variables.

• Recall that dividing two integers results in integer division,
in which any fractional part of the calculation is lost (i.e.,
\textit{truncated}).

• In the following statement the division occurs \textit{first}—the
result’s fractional part is lost before it’s assigned to
\texttt{average}:
  • \texttt{average} = \texttt{total} / \texttt{gradeCounter};
To perform a floating-point calculation with integers, create temporary floating-point values.

Unary cast operator accomplishes this task.

The cast operation `static_cast<double>(total)` creates a temporary floating-point copy of its operand in parentheses.

– Known as explicit conversion.
– The value stored in total is still an integer.

An alternative cast operation: `(double)(total)`
The calculation now consists of a floating-point value divided by the integer `gradeCounter`.

- The compiler knows how to evaluate only expressions in which the operand types of are identical.
- Compiler performs **promotion** (also called **implicit conversion**) on selected operands.
- In an expression containing values of data types `int` and `double`, C++ promotes `int` operands to `double` values.

- Cast operators are available for use with every data type and with class types as well.
Consider the following problem statement:

- A college offers a course that prepares students for the state licensing exam for real estate brokers. Last year, ten of the students who completed this course took the exam. The college wants to know how well its students did on the exam. You’ve been asked to write a program to summarize the results. You’ve been given a list of these 10 students. Next to each name is written a 1 if the student passed the exam or a 2 if the student failed.

- Your program should analyze the results of the exam as follows:
  - 1. Input each test result (i.e., a 1 or a 2). Display the prompting message “Enter result” each time the program requests another test result.
  - 2. Count the number of test results of each type.
  - 3. Display a summary of the test results indicating the number of students who passed and the number who failed.
  - 4. If more than eight students passed the exam, print the message “Bonus to instructor!”
• After reading the problem statement carefully, we make the following observations:
  – Must process test results for 10 students. A counter-controlled loop can be used because the number of test results is known in advance.
  – Each test result is a number—either a 1 or a 2. Each time the program reads a test result, the program must determine whether the number is a 1 or a 2. We test for a 1 in our algorithm. If the number is not a 1, we assume that it’s a 2. (Exercise 4.20 considers the consequences of this assumption.)
  – Two counters keep track of the exam results—one to count the number of students who passed and one to count the number of students who failed.
  – After the program has processed all the results, it must decide whether more than eight students passed the exam.
4.10 Formulating Algorithms: Nested Control Statements (cont.)

1. Initialize passes to zero
2. Initialize failures to zero
3. Initialize student counter to one
4. While student counter is less than or equal to 10
   5. Prompt the user to enter the next exam result
   6. Input the next exam result
   7. If the student passed
      8. Add one to passes
      9. Else
      10. Add one to failures
   11. Add one to student counter
12. Print the number of passes
13. Print the number of failures
14. If more than eight students passed
15. Print “Bonus to instructor!”

Fig. 4.15 | Pseudocode for examination-results problem.
// Fig. 4.16: fig04_16.cpp
// Examination-results problem: Nested control statements.
#include <iostream>
using namespace std;

int main()
{
    // initializing variables in declarations
    int passes = 0; // number of passes
    int failures = 0; // number of failures
    int studentCounter = 1; // student counter
    int result; // one exam result (1 = pass, 2 = fail)

    // process 10 students using counter-controlled loop
    while (studentCounter <= 10)
    {
        // prompt user for input and obtain value from user
        cout << "Enter result (1 = pass, 2 = fail): ";
        cin >> result; // input result

        // process result
    }

    // print results
}

Fig. 4.16 | Examination-results problem: Nested control statements. (Part I of 4.)
```cpp
// if...else nested in while
if ( result == 1 )  // if result is 1,
    passes = passes + 1;  // increment passes;
else  // else result is not 1, so
    failures = failures + 1;  // increment failures

// increment studentCounter so loop eventually terminates
studentCounter = studentCounter + 1;
}
} // end while

// termination phase; display number of passes and failures
cout << "Passed " << passes << "\nFailed " << failures << endl;

// determine whether more than eight students passed
if ( passes > 8 )
    cout << "Bonus to instructor!" << endl;
} // end main
```

**Fig. 4.16**  Examination-results problem: Nested control statements. (Part 2 of 4.)
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed 9
Failed 1
Bonus to instructor!

Fig. 4.16 | Examination-results problem: Nested control statements. (Part 3 of 4.)
• C++ allows variable initialization to be incorporated into declarations.

• The if…else statement (lines 22–25) for processing each result is nested in the while statement.

• The if statement in lines 35–36 determines whether more than eight students passed the exam and, if so, outputs the message "Bonus to instructor!".
C++ provides several assignment operators for abbreviating assignment expressions.

The `+=` operator adds the value of the expression on the right of the operator to the value of the variable on the left of the operator and stores the result in the variable on the left of the operator.

Any statement of the form

- `variable = variable operator expression;`

in which the same `variable` appears on both sides of the assignment operator and `operator` is one of the binary operators `+`, `-`, `*`, `/`, or `%` (or others we’ll discuss later in the text), can be written in the form

- `variable operator= expression;`

Thus the assignment `c += 3` adds 3 to `c`.

Figure 4.17 shows the arithmetic assignment operators, sample expressions using these operators and explanations.
4.10 Formulating Algorithms: Nested Control Statements (cont.)

<table>
<thead>
<tr>
<th>Assignment operator</th>
<th>Sample expression</th>
<th>Explanation</th>
<th>Assigns</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>c = 3, d = 5, e = 4, f = 6, g = 12;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+=</td>
<td>c += 7</td>
<td>c = c + 7</td>
<td>10 to c</td>
</tr>
<tr>
<td>-=</td>
<td>d -= 4</td>
<td>d = d - 4</td>
<td>1 to d</td>
</tr>
<tr>
<td>*=</td>
<td>e *= 5</td>
<td>e = e * 5</td>
<td>20 to e</td>
</tr>
<tr>
<td>/=</td>
<td>f /= 3</td>
<td>f = f / 3</td>
<td>2 to f</td>
</tr>
<tr>
<td>%=</td>
<td>g %= 9</td>
<td>g = g % 9</td>
<td>3 to g</td>
</tr>
</tbody>
</table>

**Fig. 4.17** | Arithmetic assignment operators.
• C++ also provides two unary operators for adding 1 to or subtracting 1 from the value of a numeric variable.

• These are the unary **increment operator**, `++`, and the unary **decrement operator**, `--`, which are summarized in Fig. 4.18.
## 4.10 Formulating Algorithms: Nested Control Statements (cont.)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Called</th>
<th>Sample expression</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>preincrement</td>
<td>++a</td>
<td>Increment a by 1, then use the new value of a in the expression in which a resides.</td>
</tr>
<tr>
<td>++</td>
<td>postincrement</td>
<td>a++</td>
<td>Use the current value of a in the expression in which a resides, then increment a by 1.</td>
</tr>
<tr>
<td>--</td>
<td>predecrement</td>
<td>--b</td>
<td>Decrement b by 1, then use the new value of b in the expression in which b resides.</td>
</tr>
<tr>
<td>--</td>
<td>postdecrement</td>
<td>b--</td>
<td>Use the current value of b in the expression in which b resides, then decrement b by 1.</td>
</tr>
</tbody>
</table>

**Fig. 4.18** | Increment and decrement operators.
4.10 Formulating Algorithms: Nested Control Statements (cont.)

**Good Programming Practice 4.9**

Unlike binary operators, the unary increment and decrement operators should be placed next to their operands, with no intervening spaces.
```cpp
1  // Fig. 4.19: fig04_19.cpp
2  // Preincrementing and postincrementing.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8      int c;
9
10     // demonstrate postincrement
11     c = 5; // assign 5 to c
12     cout << c << endl; // print 5
13     cout << c++ << endl; // print 5 then postincrement
14     cout << c << endl; // print 6
15
16     cout << endl; // skip a line
17
18     // demonstrate preincrement
19     c = 5; // assign 5 to c
20     cout << c << endl; // print 5
21     cout << ++c << endl; // preincrement then print 6
22     cout << c << endl; // print 6
23  } // end main
```

**Fig. 4.19**  |  Preincrementing and postincrementing. (Part I of 2.)
Fig. 4.19  |  Preincrementing and postincrementing. (Part 2 of 2.)
• When you increment (++) or decrement (--) a variable in a statement by itself, the preincrement and postincrement forms have the same effect, and the predecrement and postdecrement forms have the same effect.

• It’s only when a variable appears in the context of a larger expression that preincrementing the variable and postincrementing the variable have different effects (and similarly for predecrementing and post-decrementing).

• Figure 4.20 shows the precedence and associativity of the operators introduced to this point.
Common Programming Error 4.14

Attempting to use the increment or decrement operator on an expression other than a modifiable variable name or reference, e.g., writing \( ++(x + 1) \), is a syntax error.
Counter-controlled repetition requires

- the name of a control variable (or loop counter)
- the initial value of the control variable
- the loop-continuation condition that tests for the final value of the control variable (i.e., whether looping should continue)
- the increment (or decrement) by which the control variable is modified each time through the loop.

In C++, it’s more precise to call a declaration that also reserves memory a definition.
5.2 Essentials of Counter-Controlled Repetition

```cpp
// Fig. 5.1: fig05_01.cpp
// Counter-controlled repetition.
#include <iostream>
using namespace std;

int main()
{
    int counter = 1; // declare and initialize control variable
    while (counter <= 10) // loop-continuation condition
    {
        cout << counter << " ";
        counter++; // increment control variable by 1
    } // end while
    cout << endl; // output a newline
} // end main
```

**Fig. 5.1** | Counter-controlled repetition.
Common Programming Error 5.1

Floating-point values are approximate, so controlling counting loops with floating-point variables can result in imprecise counter values and inaccurate tests for termination.

Error-Prevention Tip 5.1

Control counting loops with integer values.
• The for repetition statement specifies the counter-controlled repetition details in a single line of code.
• The initialization occurs once when the loop is encountered.
• The condition is tested next and each time the body completes.
• The body executes if the condition is true.
• The increment occurs after the body executes.
• Then, the condition is tested again.
• If there is more than one statement in the body of the for, braces are required to enclose the body of the loop.
5.3 for Repetition Statement

// Fig. 5.2: fig05_02.cpp
// Counter-controlled repetition with the for statement.
#include <iostream>
using namespace std;

int main()
{
    // for statement header includes initialization,
    // loop-continuation condition and increment.
    for (int counter = 1; counter <= 10; counter++)
    {
        cout << counter << " ";
    }
    cout << endl; // output a newline
}

Fig. 5.2 Counter-controlled repetition with the for statement.
5.3 for Repetition Statement

**Fig. 5.3** | for statement header components.
The general form of the `for` statement is

```
for ( initialization; loopContinuationCondition; increment )
  statement
```

where the `initialization` expression initializes the loop’s control variable, `loopContinuationCondition` determines whether the loop should continue executing and `increment` increments the control variable.

In most cases, the `for` statement can be represented by an equivalent `while` statement, as follows:

```
initialization;
while ( loopContinuationCondition )
{
  statement
  increment;
}
```
• If the initialization expression declares the control variable, the control variable can be used only in the body of the `for` statement—the control variable will be unknown outside the `for` statement.

• This restricted use of the control variable name is known as the variable’s scope.

• The scope of a variable specifies where it can be used in a program.
5.3 for Repetition Statement (cont.)

Common Programming Error 5.3

When the control variable is declared in the initialization section of the `for` statement, using the control variable after the body is a compilation error.
The three expressions in the `for` statement header are optional (but the two semicolon separators are required).

If the `loopContinuationCondition` is omitted, C++ assumes that the condition is true, thus creating an infinite loop.

One might omit the `initialization expression if the control variable is initialized earlier in the program`.

One might omit the `increment expression if the increment is calculated by statements in the body of the `for` or if no increment is needed`. 
• The increment expression in the `for` statement acts as a stand-alone statement at the end of the body of the `for`.

• The expressions
  
  - `counter = counter + 1`
  - `counter += 1`
  - `++counter`
  - `counter++`

• are all equivalent in the incrementing portion of the `for` statement’s header (when no other code appears there).
Common Programming Error 5.5

Placing a semicolon immediately to the right of the right parenthesis of a for header makes the body of that for statement an empty statement. This is usually a logic error.
The initialization, loop-continuation condition and increment expressions of a `for` statement can contain arithmetic expressions.

The “increment” of a `for` statement can be negative, in which case the loop actually counts downward.

If the loop-continuation condition is initially false, the body of the `for` statement is not performed.
Error-Prevention Tip 5.2
Although the value of the control variable can be changed in the body of a `for` statement, avoid doing so, because this practice can lead to subtle logic errors.
• Vary the control variable from 1 to 100 in increments of 1.
  • for ( int i = 1; i <= 100; i++ )
• Vary the control variable from 100 down to 1 in decrements of 1.
  • for ( int i = 100; i >= 1; i-- )
• Vary the control variable from 7 to 77 in steps of 7.
  • for ( int i = 7; i <= 77; i += 7 )
• Vary the control variable from 20 down to 2 in steps of −2.
  • for ( int i = 20; i >= 2; i -= 2 )
• Vary the control variable over the following sequence of values: 2, 5, 8, 11, 14, 17.
  • for ( int i = 2; i <= 17; i += 3 )
• Vary the control variable over the following sequence of values: 99, 88, 77, 66, 55.
  • for ( int i = 99; i >= 55; i -= 11 )
5.4 Examples Using the for Statement

```cpp
// Fig. 5.5: fig05_05.cpp
// Summing integers with the for statement.
#include <iostream>
using namespace std;

int main()
{
    int total = 0; // initialize total
    // total even integers from 2 through 20
    for (int number = 2; number <= 20; number += 2 )
    {
        total += number;
    }
    cout << "Sum is " << total << endl; // display results
} // end main
```

Sum is 110

**Fig. 5.5** Summing integers with the for statement.
5.5 do...while Repetition Statement

- Similar to the while statement.
- The do...while statement tests the loop-continuation condition after the loop body executes; therefore, the loop body always executes at least once.
- It’s not necessary to use braces in the do...while statement if there is only one statement in the body.
  - Most programmers include the braces to avoid confusion between the while and do...while statements.
- Must end a do...while statement with a semicolon.
5.5 do...while Repetition Statement

```cpp
// Fig. 5.7: fig05_07.cpp
// do...while repetition statement.
#include <iostream>
using namespace std;

int main()
{
    int counter = 1; // initialize counter

    do
    {
        cout << counter << " "; // display counter
        counter++; // increment counter
    } while ( counter <= 10 ); // end do...while

    cout << endl; // output a newline
}
```

**Fig. 5.7** | do...while repetition statement.
The `switch multiple-selection` statement performs many different actions based on the possible values of a variable or expression.

Each action is associated with the value of a constant integral expression (i.e., any combination of character and integer constants that evaluates to a constant integer value).
5.7 switch Multiple-Selection Statement

```cpp
// Fig. 5.9: GradeBook.h
// Definition of class GradeBook that counts A, B, C, D and F grades.
// Member functions are defined in GradeBook.cpp
#include <string> // program uses C++ standard string class
using namespace std;

// GradeBook class definition
class GradeBook
{
public:
  GradeBook( string ); // constructor initializes course name
  void setCourseName( string ); // function to set the course name
  string getCourseName(); // function to retrieve the course name
  void displayMessage(); // display a welcome message
  void inputGrades(); // input arbitrary number of grades from user
  void displayGradeReport(); // display a report based on the grades

private:
  string courseName; // course name for this GradeBook
  int aCount; // count of A grades
  int bCount; // count of B grades
  int cCount; // count of C grades
  int dCount; // count of D grades
  int fCount; // count of F grades
}; // end class GradeBook
```

Fig. 5.9 GradeBook class definition.
5.7 switch Multiple-Selection Statement

```cpp
1 // Fig. 5.10: GradeBook.cpp
2 // Member-function definitions for class GradeBook that
3 // uses a switch statement to count A, B, C, D and F grades.
4 #include <iostream>
5 #include "GradeBook.h" // include definition of class GradeBook
6 using namespace std;
7 
8 // constructor initializes courseName with string supplied as argument;
9 // initializes counter data members to 0
10 GradeBook::GradeBook( string name )
11 {
12     setCourseName( name ); // validate and store courseName
13     aCount = 0; // initialize count of A grades to 0
14     bCount = 0; // initialize count of B grades to 0
15     cCount = 0; // initialize count of C grades to 0
16     dCount = 0; // initialize count of D grades to 0
17     fCount = 0; // initialize count of F grades to 0
18 } // end GradeBook constructor
```

**Fig. 5.10** GradeBook class uses switch statement to count letter grades. (Part 1 of 6.)
39    // display a welcome message to the GradeBook user
40    void GradeBook::displayMessage()
41    {
42       // this statement calls getCourseName to get the
43       // name of the course this GradeBook represents
44       cout << "Welcome to the grade book for\n" << getCourseName() << "!\n"
45            << endl;
46    } // end function displayMessage
47
48    // input arbitrary number of grades from user; update grade counter
49    void GradeBook::inputGrades()
50    {
51       int grade; // grade entered by user
52
53       cout << "Enter the letter grades." << endl
54            << "Enter the EOF character to end input." << endl;
55
56       // loop until user types end-of-file key sequence
57       while ( ( grade = cin.get() ) != EOF )
58       {

---

**Fig. 5.10** GradeBook class uses switch statement to count letter grades. (Part 3 of 6.)
59  // determine which grade was entered
60  switch ( grade ) // switch statement nested in while
61  {
62      case 'A': // grade was uppercase A
63      case 'a': // or lowercase a
64          aCount++; // increment aCount
65          break; // necessary to exit switch
66
67      case 'B': // grade was uppercase B
68      case 'b': // or lowercase b
69          bCount++; // increment bCount
70          break; // exit switch
71
72      case 'C': // grade was uppercase C
73      case 'c': // or lowercase c
74          cCount++; // increment cCount
75          break; // exit switch
76
77      case 'D': // grade was uppercase D
78      case 'd': // or lowercase d
79          dCount++; // increment dCount
80          break; // exit switch

**Fig. 5.10** GradeBook class uses `switch` statement to count letter grades. (Part 4 of 6.)
case 'F': // grade was uppercase F
    case 'f': // or lowercase f
        fCount++; // increment fCount
        break; // exit switch
    case '\n': // ignore newlines,
    case '\t': // tabs,
    case ' ': // and spaces in input
        break; // exit switch
    default: // catch all other characters
        cout << "Incorrect letter grade entered."
            << " Enter a new grade." << endl;
        break; // optional; will exit switch anyway
    } // end switch
} // end while
} // end function inputGrades

**Fig. 5.10** GradeBook class uses switch statement to count letter grades. (Part 5 of 6.)
5.7 switch Multiple-Selection Statement

```cpp
100 // display a report based on the grades entered by user
101 void GradeBook::displayGradeReport()
102 {
103     // output summary of results
104     cout << "\n\nNumber of students who received each letter grade:" << endl;
105     << "\nA: " << aCount // display number of A grades
106     << "\nB: " << bCount // display number of B grades
107     << "\nC: " << cCount // display number of C grades
108     << "\nD: " << dCount // display number of D grades
109     << "\nF: " << fCount // display number of F grades
110     << endl;
111 } // end function displayGradeReport
```

**Fig. 5.10** GradeBook class uses switch statement to count letter grades. (Part 6 of 6.)
// Fig. 5.11: fig05_11.cpp
// Create GradeBook object, input grades and display grade report.
#include "GradeBook.h" // include definition of class GradeBook

int main()
{
    // create GradeBook object
    GradeBook myGradeBook( "CS101 C++ Programming" );

    myGradeBook.displayMessage(); // display welcome message
    myGradeBook.inputGrades();  // read grades from user
    myGradeBook.displayGradeReport(); // display report based on grades
}

Welcome to the grade book for
CS101 C++ Programming!

Enter the letter grades.
Enter the EOF character to end input.
a
B

c

Fig. 5.11  |  Creating a GradeBook object and calling its member functions. (Part 1 of 2.)
Incorrect letter grade entered. Enter a new grade.

<table>
<thead>
<tr>
<th>Grade</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
</tr>
</tbody>
</table>

**Fig. 5.11** | Creating a GradeBook object and calling its member functions. (Part 2 of 2.)
• The `cin.get()` function reads one character from the keyboard.
• Normally, characters are stored in variables of type `char`; however, characters can be stored in any integer data type, because types `short`, `int` and `long` are guaranteed to be at least as big as type `char`.
• Can treat a character either as an integer or as a character, depending on its use.
• For example, the statement
  ```cpp
  cout << "The character (" << 'a' << ") has the value " << static_cast<int>('a') << endl;
  ```
• prints the character `a` and its integer value as follows:
  • The character (a) has the value 97
• The integer 97 is the character’s numerical representation in the computer.
• Generally, assignment statements have the value that is assigned to the variable on the left side of the =.

• EOF stands for “end-of-file”. Commonly used as a sentinel value.
  – *However, you do not type the value –1, nor do you type the letters EOF as the sentinel value.*
  – You type a system-dependent keystroke combination that means “end-of-file” to indicate that you have no more data to enter.

• EOF is a symbolic integer constant defined in the `<iostream>` header file.
Portability Tip 5.1
The keystroke combinations for entering end-of-file are system dependent.

Portability Tip 5.2
Testing for the symbolic constant EOF rather than -1 makes programs more portable. The ANSI/ISO C standard, from which C++ adopts the definition of EOF, states that EOF is a negative integral value, so EOF could have different values on different systems.
The `switch` statement consists of a series of `case` labels and an optional `default` case.

When the flow of control reaches the `switch`, the program evaluates the expression in the parentheses.

- The controlling expression.

The `switch` statement compares the value of the controlling expression with each `case` label.

If a match occurs, the program executes the statements for that `case`.

The `break` statement causes program control to proceed with the first statement after the `switch`. 
• Listing cases consecutively with no statements between them enables the cases to perform the same set of statements.

• Each case can have multiple statements.
  – The switch selection statement does not require braces around multiple statements in each case.

• Without break statements, each time a match occurs in the switch, the statements for that case and subsequent cases execute until a break statement or the end of the switch is encountered.
  – Referred to as “falling through” to the statements in subsequent cases.
5.8 switch Multiple-Selection Statement (cont.)

Common Programming Error 5.8

*Forgetting a break statement when one is needed in a switch statement is a logic error.*
Common Programming Error 5.9

Omitting the space between the word case and the integral value being tested in a switch statement—e.g., writing case3: instead of case 3:—is a logic error. The switch statement will not perform the appropriate actions when the controlling expression has a value of 3.
• If no match occurs between the controlling expression’s value and a `case` label, the `default` case executes.

• If no match occurs in a `switch` statement that does not contain a `default` case, program control continues with the first statement after the `switch`.
5.8 switch Multiple-Selection Statement (cont.)

Common Programming Error 5.11

Specifying a nonconstant integral expression in a switch's case label is a syntax error.
• C++ has flexible data type sizes (see Appendix C, Fundamental Types).
• C++ provides several integer types.
• The range of integer values for each type depends on the particular computer’s hardware.
• In addition to the types int and char, C++ provides the types short (an abbreviation of short int) and long (an abbreviation of long int).
• The minimum range of values for short integers is –32,768 to 32,767.
• For the vast majority of integer calculations, long integers are sufficient.
• The minimum range of values for long integers is –2,147,483,648 to 2,147,483,647.
• On most computers, *ints* are equivalent either to *short* or to *long*.

• The range of values for an *int* is at least the same as that for *short* integers and no larger than that for *long* integers.

• The data type *char* can be used to represent any of the characters in the computer’s character set.

• It also can be used to represent small integers.
Portability Tip 5.3

Because `ints` can vary in size between systems, use `long` integers if you expect to process integers outside the range –32,768 to 32,767 and you’d like to run the program on several different computer systems.
Performance Tip 5.3

If memory is at a premium, it might be desirable to use smaller integer sizes.