Lecture 8:
Control Statements (cont)

Ioan Raicu
Department of Electrical Engineering & Computer Science
Northwestern University

EECS 211
Fundamentals of Computer Programming II
April 9th, 2010
• **if...else** double-selection statement
  – specifies an action to perform when the condition is true and a different action to perform when the condition is **false**.

• The following pseudocode prints “Passed” if the student’s grade is greater than or equal to 60, or “Failed” if the student’s grade is less than 60.

  ```
  If student’s grade is greater than or equal to 60
      Print “Passed”
  Else
      Print “Failed”
  ```

• In either case, after printing occurs, the next pseudocode statement in sequence is “performed.”

• The preceding pseudocode **If...Else** statement can be written in C++ as

  ```
  if ( grade >= 60 )
      cout << "Passed";
  else
      cout << "Failed";
  ```
Fig. 4.5 | if...else double-selection statement activity diagram.
Nested if...else statements test for multiple cases by placing if...else selection statements inside other if...else selection statements.

- If student’s grade is greater than or equal to 90
  Print “A”
Else
  If student’s grade is greater than or equal to 80
    Print “B”
  Else
    If student’s grade is greater than or equal to 70
      Print “C”
    Else
      If student’s grade is greater than or equal to 60
        Print “D”
      Else
        Print “F”
This pseudocode can be written in C++ as:

```cpp
if ( studentGrade >= 90 ) // 90 and above gets "A"
    cout << "A";
else
    if ( studentGrade >= 80 ) // 80-89 gets "B"
        cout << "B";
    else
        if ( studentGrade >= 70 ) // 70-79 gets "C"
            cout << "C";
        else
            if ( studentGrade >= 60 ) // 60-69 gets "D"
                cout << "D";
            else // less than 60 gets "F"
                cout << "F";
```

If `studentGrade` is greater than or equal to 90, the first four conditions are true, but only the output statement after the first test executes. Then, the program skips the `else`-part of the “outermost” `if...else` statement.
Most write the preceding `if...else` statement as:

```cpp
if ( studentGrade >= 90 ) // 90 and above gets "A"
    cout << "A";
else if ( studentGrade >= 80 ) // 80-89 gets "B"
    cout << "B";
else if ( studentGrade >= 70 ) // 70-79 gets "C"
    cout << "C";
else if ( studentGrade >= 60 ) // 60-69 gets "D"
    cout << "D";
else // less than 60 gets "F"
    cout << "F";
```

The two forms are identical except for the spacing and indentation, which the compiler ignores.

The latter form is popular because it avoids deep indentation of the code to the right, which can force lines to wrap.
• The C++ compiler always associates an `else` with the immediately preceding `if` unless told to do otherwise by the placement of braces (`{` and `}`).

• This behavior can lead to what’s referred to as the **dangling-else** problem.

  ```cpp
  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.
• The compiler actually interprets the statement as
  
  ```
  if ( x > 5 )
      if ( y > 5 )
          cout << "x and y are > 5";
      else
          cout << "x is <= 5";
  ```

• To force the nested if...else statement to execute as intended, use:
  
  ```
  if ( x > 5 )
      {
          if ( y > 5 )
              cout << "x and y are > 5";
      }
  else
      cout << "x is <= 5";
  ```

• Braces ({}) indicate that the second if statement is in the body of the first if and that the else is associated with the first if.
• Just as a block can be placed anywhere a single statement can be placed, it’s also possible to have no statement at all—called a null statement (or an empty statement).

• The null statement is represented by placing a semicolon (;) where a statement would normally be.
A repetition statement (also called a looping statement or a loop) allows you to specify that a program should repeat an action while some condition remains true.

- While there are more items on my shopping list
  - Purchase next item and cross it off my list

“There are more items on my shopping list” is true or false.

- If true, “Purchase next item and cross it off my list” is performed.
  - Performed repeatedly while the condition remains true.
- The statement contained in the While repetition statement constitutes the body of the While, which can be a single statement or a block.
- Eventually, the condition will become false, the repetition will terminate, and the first pseudocode statement after the repetition statement will execute.
• Consider a program segment designed to find the first power of 3 larger than 100. Suppose the integer variable `product` has been initialized to 3.

• When the following `while` repetition statement finishes executing, `product` contains the result:
  ```
  int product = 3;

  while ( product <= 100 )
    product = 3 * product;
  ```
Common Programming Error 4.5

Not providing, in the body of a `while` statement, an action that eventually causes the condition in the `while` to become false normally results in a logic error called an infinite loop, in which the repetition statement never terminates. This can make a program appear to “hang” or “freeze” if the loop body does not contain statements that interact with the user.
4.7 while Repetition Statement (cont.)

Fig. 4.6 | while repetition statement UML activity diagram.

Corresponding C++ statement:
product = 3 * product;
• Consider the following problem statement:
  – A class of ten students took a quiz. The grades (integers in the range 0 to 100) for this quiz are available to you. Calculate and display the total of all student grades and the class average on the quiz.

• The class average is equal to the sum of the grades divided by the number of students.

• The algorithm for solving this problem on a computer must input each of the grades, calculate the average and print the result.
• We use counter-controlled repetition to input the grades one at a time.
  – This technique uses a variable called a counter to control the number of times a group of statements will execute (also known as the number of iterations of the loop).
  – Often called definite repetition because the number of repetitions is known before the loop begins executing.
Software Engineering Observation 4.3

Experience has shown that the most difficult part of solving a problem on a computer is developing the algorithm for the solution. The process of producing a working C++ program from the algorithm is typically straightforward.
4.8 **Formulating Algorithms: Counter-Controlled Repetition (cont.)**

1. Set total to zero
2. Set grade counter to one
3. While grade counter is less than or equal to ten
4. Prompt the user to enter the next grade
5. Input the next grade
6. Add the grade into the total
7. Add one to the grade counter
8. Set the class average to the total divided by ten
9. Print the total of the grades for all students in the class
10. Print the class average

**Fig. 4.7** Pseudocode for solving the class average problem with counter-controlled repetition.
• A **total** is a variable used to accumulate the sum of several values.

• A **counter** is a variable used to count—in this case, the grade counter indicates which of the 10 grades is about to be entered by the user.

• Variables used to store totals are normally initialized to zero before being used in a program; otherwise, the sum would include the previous value stored in the total’s memory location.
4.8 Formulating Algorithms: Counter-Controlled Repetition (cont.)

```cpp
34 // display a welcome message to the GradeBook user
35 void GradeBook::displayMessage()
36 {
37    cout << "Welcome to the grade book for\n" << getCourseName() << "!\n"
38    << endl;
39 } // end function displayMessage
40
41 // determine class average based on 10 grades entered by user
42 void GradeBook::determineClassAverage()
43 {
44    int total; // sum of grades entered by user
45    int gradeCounter; // number of the grade to be entered next
46    int grade; // grade value entered by user
47    int average; // average of grades
48
49    // initialization phase
50    total = 0; // initialize total
51    gradeCounter = 1; // initialize loop counter
```

**Fig. 4.9** | Class average problem using counter-controlled repetition: GradeBook source code file. (Part 3 of 4.)
// processing phase
while ( gradeCounter <= 10 ) // loop 10 times
{
    cout << "Enter grade: "; // prompt for input
    cin >> grade; // input next grade
    total = total + grade; // add grade to total
    gradeCounter = gradeCounter + 1; // increment counter by 1
}
// end while

// termination phase
average = total / 10; // integer division yields integer result

// display total and average of grades
cout << "\nTotal of all 10 grades is " << total << endl;
cout << "Class average is " << average << endl;
} // end function determineClassAverage

Fig. 4.9 | Class average problem using counter-controlled repetition: GradeBook
source code file. (Part 4 of 4.)
Counter variables are normally initialized to zero or one, depending on their use.

An uninitialized variable contains a “garbage” value (also called an undefined value)—the value last stored in the memory location reserved for that variable.

The variables grade and average (for the user input and calculated average, respectively) need not be initialized before they’re used—their values will be assigned as they’re input or calculated later in the function.
Common Programming Error 4.6

Not initializing counters and totals can lead to logic errors.
Error-Prevention Tip 4.2

Initialize each counter and total, either in its declaration or in an assignment statement. Totals are normally initialized to 0. Counters are normally initialized to 0 or 1, depending on how they’re used.
4.8 Formulating Algorithms: Counter-Controlled Repetition (cont.)

Good Programming Practice 4.7

Declare each variable on a separate line with its own comment for readability.
The averaging calculation performed in response to the function call in line 12 of Fig. 4.10 produces an integer result.

Dividing two integers results in integer division—any fractional part of the calculation is lost (i.e., truncated).
4.8 Formulating Algorithms: Counter-Controlled Repetition (cont.)

Common Programming Error 4.7

Assuming that integer division rounds (rather than truncates) can lead to incorrect results. For example, $7 \div 4$, which yields 1.75 in conventional arithmetic, truncates to 1 in integer arithmetic, rather than rounding to 2.
Let’s generalize the class average problem.
- Develop a class average program that processes grades for an arbitrary number of students each time it’s run.

The program must process an arbitrary number of grades.
- How can the program determine when to stop the input of grades?

Can use a special value called a sentinel value (also called a signal value, a dummy value or a flag value) to indicate “end of data entry.”

Sentinel-controlled repetition is often called indefinite repetition
- the number of repetitions is not known in advance.

The sentinel value must not be an acceptable input value.
4.9 Formulating Algorithms: Sentinel-Controlled Repetition

Common Programming Error 4.9

Choosing a sentinel value that is also a legitimate data value is a logic error.
4.9 Formulating Algorithms: Sentinel-Controlled Repetition

1. Initialize total to zero
2. Initialize counter to zero
3.
4. Prompt the user to enter the first grade
5. Input the first grade (possibly the sentinel)
6.
7. While the user has not yet entered the sentinel
   8.   Add this grade into the running total
   9.   Add one to the grade counter
10.  Prompt the user to enter the next grade
11.  Input the next grade (possibly the sentinel)
12.
13. If the counter is not equal to zero
14.   Set the average to the total divided by the counter
15.   Print the total of the grades for all students in the class
16.   Print the class average
17.   else
18.   Print “No grades were entered”

Fig. 4.11 | Class average problem pseudocode algorithm with sentinel-controlled repetition.
Common Programming Error 4.10
An attempt to divide by zero normally causes a fatal runtime error.

Error-Prevention Tip 4.3
When performing division by an expression whose value could be zero, explicitly test for this possibility and handle it appropriately in your program (such as by printing an error message) rather than allowing the fatal error to occur.
An averaging calculation is likely to produce a number with a decimal point—a real number or floating-point number (e.g., 7.33, 0.0975 or 1000.12345).

Type int cannot represent such a number.

C++ provides several data types for storing floating-point numbers in memory, including float and double.

Compared to float variables, double variables can typically store numbers with larger magnitude and finer detail—more digits to the right of the decimal point—also known as the number’s precision.

Cast operator can be used to force the averaging calculation to produce a floating-point numeric result.
// Fig. 4.13: GradeBook.cpp

// Member-function definitions for class GradeBook that solves the
// class average program with sentinel-controlled repetition.

#include <iostream>
#include <iomanip> // parameterized stream manipulators
#include "GradeBook.h" // include definition of class GradeBook
using namespace std;

// constructor initializes courseName with string supplied as argument
GradeBook::GradeBook( string name )
{
    setCourseName( name ); // validate and store courseName
} // end GradeBook constructor

// function to set the course name;
// ensures that the course name has at most 25 characters
void GradeBook::setCourseName( string name )
{
    if ( name.length() <= 25 ) // if name has 25 or fewer characters
        courseName = name; // store the course name in the object
    else // if name is longer than 25 characters
        { // set courseName to first 25 characters of parameter name

Fig. 4.13 | Class average problem using sentinel-controlled repetition: GradeBook
source code file. (Part 1 of 5.)
4.9 Formulating Algorithms: Sentinel-Controlled Repetition (cont.)

```cpp
// determine class average based on 10 grades entered by user
void GradeBook::determineClassAverage()
{
  int total; // sum of grades entered by user
  int gradeCounter; // number of grades entered
  int grade; // grade value
  double average; // number with decimal point for average

  // initialization phase
  total = 0; // initialize total
  gradeCounter = 0; // initialize loop counter

  // code to read grades and calculate average
}
```

**Fig. 4.13** Class average problem using sentinel-controlled repetition: GradeBook source code file. (Part 3 of 5.)
// processing phase
// prompt for input and read grade from user
cout << "Enter grade or -1 to quit: ";
cin >> grade; // input grade or sentinel value

// loop until sentinel value read from user
while ( grade != -1 ) // while grade is not -1
{
    total = total + grade; // add grade to total
    gradeCounter = gradeCounter + 1; // increment counter

    // prompt for input and read next grade from user
    cout << "Enter grade or -1 to quit: ";
cin >> grade; // input grade or sentinel value
}
// end while

---

**Fig. 4.13** Class average problem using sentinel-controlled repetition: *GradeBook* source code file. (Part 4 of 5.)
4.9 Formulating Algorithms: Sentinel-Controlled Repetition (cont.)

```cpp
// termination phase
if ( gradeCounter != 0 ) // if user entered at least one grade...
{
    // calculate average of all grades entered
    average = static_cast< double >(( total ) / gradeCounter);

    // display total and average (with two digits of precision)
    cout << "\nTotal of all " << gradeCounter << " grades entered is "
    << total << endl;
    cout << "Class average is " << setprecision( 2 ) << fixed << average
    << endl;
} // end if
else // no grades were entered, so output appropriate message
    cout << "No grades were entered" << endl;
} // end function determineClassAverage
```

Fig. 4.13 Class average problem using sentinel-controlled repetition: GradeBook source code file. (Part 5 of 5.)
The call to `setprecision` in line 79 (with an argument of 2) indicates that `double` values should be printed with two digits of `precision` to the right of the decimal point (e.g., 92.37).

- Parameterized stream manipulator (argument in parentheses).
- Programs that use these must include the header `<iomanip>`.

`endl` is a nonparameterized stream manipulator and does not require the `<iomanip>` header file.

If the precision is not specified, floating-point values are normally output with six digits of precision.
Stream manipulator **fixed** indicates that floating-point values should be output in **fixed-point format**, as opposed to **scientific notation**.

- Fixed-point formatting is used to force a floating-point number to display a specific number of digits.
- Specifying fixed-point formatting also forces the decimal point and trailing zeros to print, even if the value is a whole number amount, such as 88.00.
  - Without the fixed-point formatting option, such a value prints in C++ as 88 without the trailing zeros and decimal point.
When the stream manipulators `fixed` and `setprecision` are used in a program, the printed value is rounded to the number of decimal positions indicated by the value passed to `setprecision` (e.g., the value 2 in line 79), although the value in memory remains unaltered.

It’s also possible to force a decimal point to appear by using stream manipulator `showpoint`.

- If `showpoint` is specified without `fixed`, then trailing zeros will not print.
- Both can be found in header `<iostream>`. 
Welcome to the grade book for
CS101 C++ Programming

Enter grade or -1 to quit: 97
Enter grade or -1 to quit: 88
Enter grade or -1 to quit: 72
Enter grade or -1 to quit: -1

Total of all 3 grades entered is 257
Class average is 85.67

Fig. 4.14  |  Class average problem using sentinel-controlled repetition: Creating a
GradeBook object and invoking its determineClassAverage member function. (Part
2 of 2.)
• Notice the block in the while loop in Fig. 4.13.
• Without the braces, the last three statements in the body of the loop would fall outside the loop, causing the computer to interpret this code incorrectly, as follows:

```cpp
// loop until sentinel value read from user
while ( grade != -1 )
    total = total + grade; // add grade to total
    gradeCounter = gradeCounter + 1; // increment counter

// prompt for input and read next grade from user
    cout << "Enter grade or -1 to quit: ";
    cin >> grade;
```

• This would cause an infinite loop in the program if the user did not input −1 for the first grade (in line 57).
Common Programming Error 4.11

Omitting the braces that delimit a block can lead to logic errors, such as infinite loops. To prevent this problem, some programmers enclose the body of every control statement in braces, even if the body contains only a single statement.
Variables of type \texttt{float} represent \textit{single-precision floating-point numbers} and have seven significant digits on most 32-bit systems.

Variables of type \texttt{double} represent \textit{double-precision floating-point numbers}.

- These require twice as much memory as \texttt{float} variables and provide 15 significant digits on most 32-bit systems.
- Approximately double the precision of \texttt{float} variables.

C++ treats all floating-point numbers in a program’s source code as \texttt{double} values by default.

- Known as \textit{floating-point constants}.

Floating-point numbers often arise as a result of division.
Common Programming Error 4.12

Using floating-point numbers in a manner that assumes they’re represented exactly (e.g., using them in comparisons for equality) can lead to incorrect results. Floating-point numbers are represented only approximately.