# Chapter 3: Control Statements and Algorithm Development, Part 1

## C++ How to Program: An Objects-Natural Approach, 11/e


Presented by
Paul Deitel, CEO, Deitel & Associates, Inc.

# Chapter 3: Control Statements, Part 1

- `if` and `if...else` selection statements
- `while` iteration statement
- Compound assignment operators
- Increment and decrement operators
- Why fundamental data types are not portable
- Objects Natural Case Study: Super-Sized Integers with Boost Multiprecision

# Chapter 3: Algorithm Development and Control Statements—Part 1


Presented by
Paul Deitel, CEO, Deitel & Associates, Inc.

# Chapter 3: Algorithm Development and Control Statements—Part 1

- Basic problem-solving techniques
- Develop algorithms using top-down, stepwise refinement
- `if` and `if...else` selection statements
- `while` iteration statement
- Counter-controlled iteration
- Sentinel-controlled iteration

# Chapter 3: Algorithm Development and Control Statements—Part 1

- Nested control statements
- Compound assignment operators
- Increment and decrement operators
- Why fundamental data types are not portable
- Objects Natural Case Study: Super-Sized Integers with Boost Multiprecision

# Algorithms

- An algorithm is a procedure for solving a problem
  - actions to execute
  - order in which these actions execute
- “Rise-and-shine” algorithm
  - (1) Get out of bed; (2) take off pajamas; (3) take a shower; (4) get dressed; (5) eat breakfast; (6) carpool to work
  - (1) Get out of bed; (2) take off pajamas; (3) get dressed; (4) take a shower; (5) eat breakfast; (6) carpool to work
- Specifying the order in which statements (actions) execute in a program is called program control

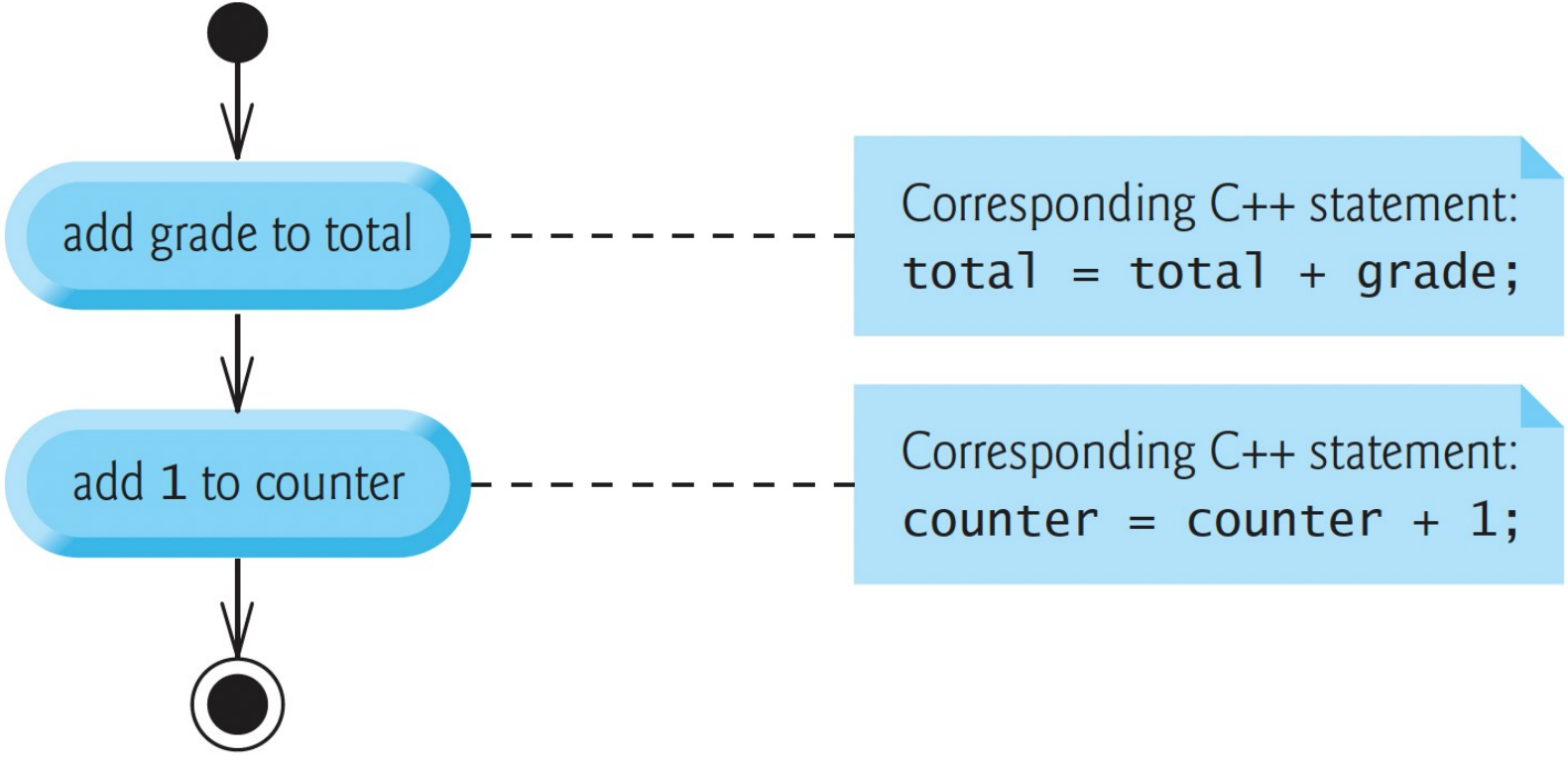
# Pseudocode

- Informal language that helps you develop algorithms without worrying about C++ language syntax
- Addition program pseudocode
  1. Prompt the user to enter the first integer
  2. Input the first integer
  3. 
  4. Prompt the user to enter the second integer
  5. Input the second integer
  6. 
  7. Add first integer and second integer, store result
  8. Display result

# Control Structures

- All programs can be written in terms of
  - Sequence
  - Selection
  - Iteration

# Sequence Structure



# Selection Statements

- `if`
- `if…else`
- `switch`

# Iteration Statements

- `while`
- `do…while`
- `for`
- range-based `for`

# Keywords (1 of 3)

**Keywords common to C and C++**

| | | | | |
|---|---|---|---|---|
| asm | auto | break | case | char |
| const | continue | default | do | double |
| else | enum | extern | float | for |
| goto | if | inline | int | long |
| register | return | short | signed | sizeof |
| static | struct | switch | typedef | union |
| unsigned | void | volatile | while | |

# Keywords (2 of 3)

**C++-only keywords**

| | | | | |
|---|---|---|---|---|
| alignas | alignof | and | and_eq | bitand |
| bitor | bool | catch | char16_t | char32_t |
| class | compl | const_cast | constexpr | decltype |
| delete | dynamic_cast | explicit | export | false |
| friend | mutable | namespace | new | noexcept |
| not | not_eq | nullptr | operator | or |
| or_eq | private | protected | public | reinterpret_cast |
| static_assert | static_cast | template | this | thread_local |
| throw | true | try | typeid | typename |
| using | virtual | wchar_t | xor | xor_eq |

# Keywords (3 of 3)

**C++20 keywords**

| | | | | |
|---|---|---|---|---|
| char8_t | concept | char16_t | consteval | constinit |
| co_await | co_return | co_yield | requires | |

**Other Special Identifiers**

- Identifiers with special meaning
  - final, import, module, override, transaction_safe, transaction_safe_dynamic
- Experimental keywords
  - atomic_cancel, atomic_commit, atomic_noexcept, reflexpr, synchronized

# `if` Single-Selection Statement

```
if (studentGrade >= 60) {
   cout << "Passed";
}
```

- Indentation of the second line of this selection statement is optional, but recommended for program clarity

# `bool` Data Type

- Condition
  - Any expression that evaluates to zero or nonzero
  - Zero is false, nonzero is true
- Data type **bool** for Boolean variables
  - Values **true** and **false**—each is a C++ keyword
- For compatibility with C
  - `true` can be represented as any nonzero value
    - Compilers typically use 1
  - `false` can be represented as `0`

# if…else Double-Selection Statement

```
if (grade >= 60) {
    cout << "Passed";
}
else {
    cout << "Failed";
}
```

# Nested if…else Statements (1 of 2)

```
if (studentGrade >= 90) {
   cout << "A";
}
else {
   if (studentGrade >= 80) {
      cout << "B";
   }
   else {
      if (studentGrade >= 70) {
         cout << "C";
      }
      else {
         if (studentGrade >= 60) {
            cout << "D";
         }
         else {
            cout << "F";
         }
      }
   }
}
```

# Nested if…else Statements (2 of 2)

```
if (studentGrade >= 90) {
   cout << "A";
}
else if (studentGrade >= 80) {
   cout << "B";
}
else if (studentGrade >= 70) {
   cout << "C";
}
else if (studentGrade >= 60) {
   cout << "D";
}
else {
   cout << "F";
}
```

# Conditional Operator (?:)

- **Conditional operator (?:)** can be used in place of an `if...else` statement
- C++'s only **ternary operator**
- `cout << (studentGrade >= 60 ? "Passed" : "Failed");`

# while Iteration Statement

```
int product{3};

while (product <= 100) {
    product = 3 * product;
}
```

# Counter-Controlled Iteration

- Problem statement
  - A class of ten students took a quiz. The grades (integers in the range 0–100) for this quiz are available to you. Determine the class average on the quiz.
- Class average is the sum of the grades divided by the number of students
- Program must
  - input each grade
  - total all the grades entered
  - perform the averaging calculation
  - print the result

# Pseudocode Algorithm with Counter-Controlled Iteration

- Use pseudocode to list the actions to execute and specify the order in which they should execute
- 10 grades: Use counter-controlled iteration to input the grades one at a time
- Use a **counter** (or **control variable**) to control the number of times a set of statements will execute
- Counter-controlled iteration is often called **definite iteration** because the number of iterations is known before the loop begins executing

# Pseudocode Algorithm with Counter-Controlled Iteration

1. set total to zero
2. set grade counter to one
3. 
4. while grade counter is less than or equal to ten
5. prompt the user to enter the next grade
6. input the next grade
7. add the grade into the total
8. add one to the grade counter
9. 
10. set the class average to the total divided by ten
11. print the class average

# Integer Division and Truncation

- Integer division produces an integer result
- 846 divided by 10, should yield the floating-point value 84.6
- Yields 84 because both operands are integers

# Sentinel-Controlled Iteration

- Problem statement
  - Develop a class-averaging program that processes grades for an arbitrary number of students each time it’s run
- Don’t know number of students
- Sentinel value—a way to indicate end of data entry
- Should be a value that is not a valid data value

# Top-Down, Stepwise Refinement

- Begin with single statement that conveys the overall function of the program—known as the **top**
  - **determine the class average for the quiz**
- Refine this as many times as you need until you're ready to convert your pseudocode to C++ code
- First refinement
  - **initialize variables**
  - **input, sum and count the quiz grades**
  - **calculate and print the class average**

# Top-Down, Stepwise Refinement

- Second refinement commits to specific variables
  - running total of the numbers
  - count of how many numbers we've processed
  - variable to receive each grade entered by the user
  - variable to hold the calculated average
- The pseudocode statement
  - **initialize variables**
- Can be refined as
  - **initialize total to zero**
  - **initialize counter to zero**

# Top-Down, Stepwise Refinement

- The pseudocode statement
  - **input, sum and count the quiz grades**
- Can be refined as
  - **prompt the user to enter the first grade**
  - **input the first grade (possibly the sentinel)**
  - 
  - **while the user has not yet entered the sentinel**
  - **add this grade into the running total**
  - **add one to the grade counter**
  - **prompt the user to enter the next grade**
  - **input the next grade (possibly the sentinel)**

# Top-Down, Stepwise Refinement

- **The pseudocode statement**
  - **calculate and print the class average**
- **Can be refined as**
  - if the counter is not equal to zero
  - set the average to the total divided by the counter
  - print the average
  - else
  - print “No grades were entered”

# Top-Down, Stepwise Refinement

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 average
16. else
17. print “No grades were entered”

# Nested Control Statements

- Problem statement
  - A college offers a course that prepares students for the state licensing exam for real-estate brokers. Last year, 10 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 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 "Enter result" on the screen 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 pass, print "Bonus to instructor!"

# Nested Control Statements

- Observations
  - The program must process test results for 10 students
  - Each time the program reads a test result, it must determine whether it’s a 1 or a 2
  - Two counters are used to keep track of the exam results—one to count the number of students who passed the exam and one to count the number who failed.
  - After the program has processed all the results, it must decide whether more than eight students passed the exam

# Top-Down, Stepwise Refinement

- Top
  - **analyze exam results and decide whether a bonus should be paid**
- First refinement
  - **initialize variables**
  - **input the 10 exam results, and count passes and failures**
  - **print a summary of the exam results and decide whether a bonus should be paid**

# Top-Down, Stepwise Refinement

- The pseudocode statement
  - **initialize variables**
- can be refined as
  - **initialize passes to zero**
  - **initialize failures to zero**
  - **initialize student counter to one**

# Top-Down, Stepwise Refinement

- The pseudocode statement
  - **input the 10 exam results, and count passes and failures**
- Can be refined as
  - **while student counter is less than or equal to 10**
  - **    prompt the user to enter the next exam result**
  - **    input the next exam result**
  - 
  - **    if the student passed**
  - **        add one to passes**
  - **    else**
  - **        add one to failures**
  - 
  - **    add one to student counter**

# Top-Down, Stepwise Refinement

- The pseudocode statement
  - **print a summary of the exam results and decide whether a bonus should be paid**
- Can be refined as
  - **print the number of passes**
  - **print the number of failures**
  - 
  - **if more than eight students passed**
  - **print “Bonus to instructor!”**

# Top-Down, Stepwise Refinement —

# Complete Second Refinement

```
1. initialize passes to zero
2. initialize failures to zero
3. initialize student counter to one
4.
5. while student counter is less than or equal to 10
6.     prompt the user to enter the next exam result
7.     input the next exam result
8.
9.     if the student passed
10.        add one to passes
11.    else
12.        add one to failures
13.
14.    add one to student counter
15.
16. print the number of passes
17. print the number of failures
18.
19. if more than eight students passed
20.    print “Bonus to instructor!”
```

# Preventing Narrowing Conversions with List Initialization

- `int studentCounter{1};`
- Also can be written as
  - `int studentCounter = 1;`
- For fundamental-type variables, list initialization prevents narrowing conversions
  - `int x = 12.7;`
  - C++ converts `12.7` to the `int` value 12
  - This narrowing conversion **loses data**

# Preventing Narrowing Conversions with List Initialization

- `int x{12.7};`
  - Compilation error
  - Even `12.0` gives an error
- **Visual Studio — conversion from 'double' to 'int' requires a narrowing conversion**
- **GNU C++ — error: narrowing conversion of '1.26e+1' from 'double' to 'int' [-Wnarrowing]**
- **clang++ — error: type 'double' cannot be narrowed to 'int' in initializer list [-Wc++11-narrowing]**

# Compound Assignment Operators

- `total = total + grade;`
- Assume: c = 3, d = 5, e = 4, f = 6, g = 12

| OPERATOR | SAMPLE EXPRESSION | EXPLANATION | ASSIGNS |
|---|---|---|---|
| += | c += 7 | c = c + 7 | 10 to c |
| -= | d -= 4 | d = d - 4 | 1 to d |
| *= | e *= 5 | e = e * 5 | 20 to e |
| /= | f /= 3 | f = f / 3 | 2 to f |
| %= | g %= 9 | g = g % 9 | 3 to g |

# Increment and Decrement Operators

- `passes = passes + 1;`

| Operator | Operator name | Sample expression | Explanation |
|---|---|---|---|
| `++` | prefix increment | `++number` | Increment `number` by 1, then use the new value of `number` in the expression in which `number` resides. |
| `++` | postfix increment | `number++` | Use the current value of `number` in the expression in which `number` resides, then increment `number` by 1. |
| `--` | prefix decrement | `--number` | Decrement `number` by 1, then use the new value of `number` in the expression in which `number` resides. |
| `--` | postfix decrement | `number--` | Use the current value of `number` in the expression in which `number` resides, then decrement `number` by 1. |

# Fundamental Types Are Not Portable

- In C and C++, an `int` might be represented by
  - 16 bits (2 bytes)
  - 32 bits (4 bytes)
  - 64 bits (8 bytes)
- Code using integers is not always portable across platforms
- Sometimes must write multiple versions of programs to use different integer types on different platforms

# Fundamental Types Are Not Portable

- Among C++'s integer types are `int`, `long` and `long long`
- C++ standard requires type `int` to be at least 16 bits, type `long` to be at least 32 bits and type `long long` to be at least 64 bits
- Also requires that an `int`'s size be less than or equal to a `long`'s size and that a `long`'s size be less than or equal to a `long long`'s size

# Objects Natural Case Study: Super-Sized Integers

- Explosive growth of Internet communications and data storage on Internet-connected devices has dramatically increased the importance of privacy and security
- Cryptography has been used for thousands of years to encode data, making it difficult (and hopefully impossible) for unauthorized users to read, which is critically important in today’s connected world

# Objects Natural Case Study: Super-Sized Integers

## Some Applications Need Integers Outside a long long Integer's Range

- Cryptography algorithms perform calculations with integers far larger than C++'s `long long` type can store
  - –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
- Maximum of 19 decimal digits (which can represent integers in the quintillions)
- RSA Public-Key Cryptography uses enormous prime numbers consisting of hundreds of digits
- A key reason why RSA is so secure is the sheer amount of time required to factor the product of massive primes

# Objects Natural Case Study: Super-Sized Integers

## Integers Outside the long long Range Require Custom Programming

- How can we manipulate huge integers with arbitrary numbers of digits?
- Custom programming with C++’s class mechanism
- Objects Natural Approach
  - check first for free, open-source class libraries to see if such a “huge-integer” class already exists

# Objects Natural Case Study: Super-Sized Integers

## Integers Outside the long long Range Require Custom Programming

- How can we manipulate huge integers with arbitrary numbers of digits?
- Custom programming with C++’s class mechanism
- Objects Natural Approach
  - check first for free, open-source class libraries to see if such a “huge-integer” class already exists

# Objects Natural Case Study: Super-Sized Integers

Huge Integers with the Boost Multiprecision Open-Source Library's `cpp_int` Class

- Many preexisting C++ open-source classes for creating and manipulating huge integers
- https://github.com/boostorg/multiprecision/
- Boost provides 168 open-source C++ libraries
- "Breeding ground" for new capabilities that often are incorporated into the C++ standard libraries