An Introduction to Programming with C++ Eighth Edition

Chapter 4: Completing the Problem-Solving Process

Objectives

- Get numeric and character data from the keyboard
- Display information on the computer screen
- Write arithmetic expressions
- Type cast a value
- Write an assignment statement
- Code the algorithm into a program
- Desk-check a program
- Evaluate and modify a program

Finishing Step 4 in the Problem-Solving Process

- The fourth step in the problem-solving process is to code algorithm into a program
- Begin by declaring a memory location for each input, processing, and output value in IPO chart
- Optionally initialize each value (highly preferred)
- Next, you code the instructions for the algorithm

Finishing Step 4 in the Problem-Solving Process (cont'd.)

Problem specification

Addison O'Reilly wants a program that calculates and displays the cost of a 4K Ultra HD TV, which is finally on sale at one of the stores in her area. The program should calculate the cost by multiplying the sale price by the state sales tax rate and then adding the result to the sale price.

IPO chart information

Input

sale price
sales tax rate

Processing

sales tax

Output

c st

Algorithm:

e ter t e sale price a sales&ax rat
calc late t e sales tax ltipl i e t e sal
price t e sales tax&rat
calc late t e c st a i t e&sales tax t t
sale price
ispla t e c st

C++ instructions

```
double salePrice = 0.0;
double taxRate = 0.0;

double salesTax = 0.0;

double cost = 0.0;
```

Figure 4-1 Problem specification, IPO chart information, and variable declaration

Getting Data from the Keyboard

- C++ uses stream objects to perform input/output operations
- A **stream** is a sequence of characters
- The **`cin`** object is used to obtain information from the keyboard (program pauses while user enters data)
- The **extraction operator** (>>) takes information out of `cin` object and stores it in internal memory
  - Syntax: `cin >>` *variableName;*

Getting Data from the Keyboard (cont'd.)

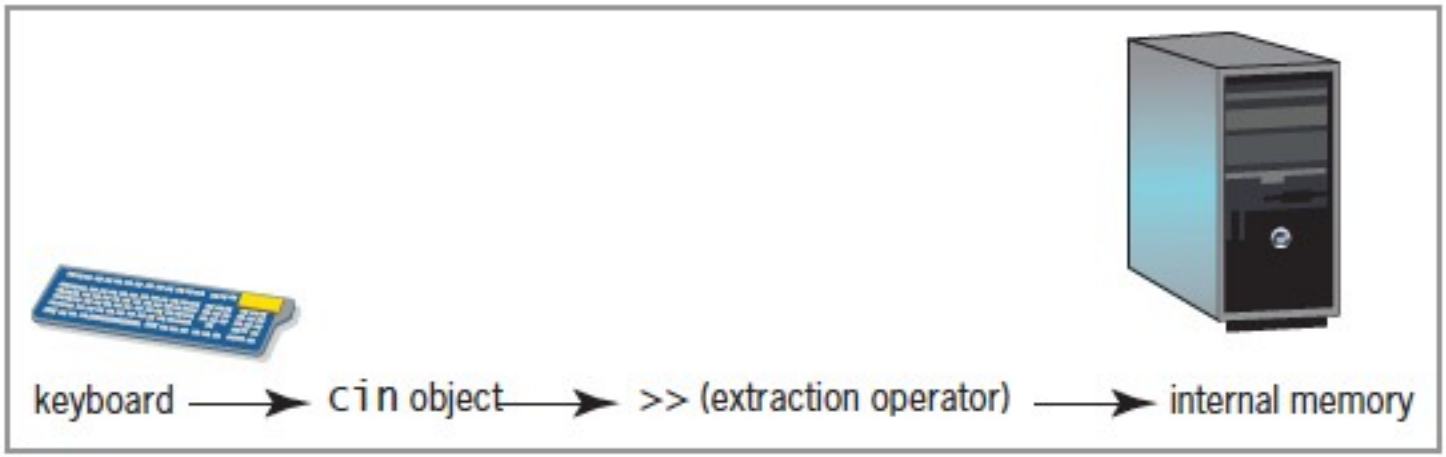

Figure 4-2 Relationship among the keyboard, `cin` object, extraction operator, and internal memory

Getting Data from the Keyboard (cont'd.)

HOW TO Use cin and >> to Get Numeric or Character Data

Syntax

```
cin >> variableName;
```

semicolon

extraction operator

Example 1

```
double price = 0.0;
cin >> price;
```

Example 2

```
char middleInitial = ' ';
cin >> middleInitial;
```

Figure 4-3 How to use `cin` and >> to get numeric or character data

Getting Data from the Keyboard (cont'd.)

IPO chart information

Input

sale price
sales tax rate

Processing

sales tax

Output

c st

Algorithm:

e ter t e sale price a sales tax rat

calc late t e sales tax ltipl i e t e sal price t e sales tax rat

calc late t e c st a i t e sales tax t t sale price

ispla t e c st

C++ instructions

```
double salePrice = 0.0;
double taxRate = 0.0;

double salesTax = 0.0;

double cost = 0.0;

cin >> salePrice;
cin >> taxRate;
```

Figure 4-4 Input statements for the Addison O'Reilly problem

Displaying Messages on the Computer Screen

- You use a prompt (message) to let the user know what data is to be entered
- The **`cout`** object is used with the **insertion operator** (<<) to display information on the screen
- Information can be any combination of literal constants, named constants, and variables
- Multiple items can be printed in the same statement
  - Syntax: `cout << item1 [<< item2 << itemN];`
  - Part in brackets is optional

Displaying Messages on the Computer Screen (cont'd.)

- A stream manipulator is used to manipulate (manage) the characters in an input or output string
- **`endl`** is a stream manipulator that advances the cursor to the next line on the screen
  - Equivalent to pressing the Enter key

    (carriage return and line feed)

Displaying Messages on the Computer Screen (cont'd.)

HOW TO Use the cout Object

Syntax

```
cout << item1 [<< item2 << itemN];
```

insertion operator

semicolon

Examples

```
cout << "Enter the price: ";
cout << "What is your middle initial? ";
cout << "End of program";
cout << "Bonus: $" << bonusAmt << endl;
```

stream manipulator

Note: The last `cout` statement is equivalent to the following three lines of code:

```
cout << "Bonus: $";
cout << bonusAmt;
cout << endl;
```

Figure 4-5 How to use the `cout` object

Displaying Messages on the Computer Screen (cont'd.)

IPO chart information	C++ instructions
Input	
sale price	`double salePrice = 0.0;`
sales tax rate	`double taxRate = 0.0;`
Processing	
sales tax	`double salesTax = 0.0;`
Output	
c st	`double cost = 0.0;`
Algorithm:	
e ter t e sale price a sales tax rat	`cout << "Enter the sale price: ";`
	`cin >> salePrice;`
	`cout << "Enter the sales tax rate: ";`
	`cin >> taxRate;`
calc late t e sales tax ltipl i t e sale price t e sales tax rate	
calc late t e c st a i t e sales tax t t e sale price	
ispla t e c st	`cout << "Cost: $" << cost << endl;`

Figure 4-6 Prompts and output statement for the Addison O'Reilly problem

Arithmetic Operators in C++

- You can evaluate arithmetic expressions in C++ using arithmetic operators
- Operators are negation (−), addition (+), subtraction (−), multiplication (*), division (/), and modulus (%)
- Negation and subtraction use the same symbol, but negation is a unary operator (one operand) and subtraction is a binary operator (two operands)
- Modulus gives remainder when dividing two integers

Arithmetic Operators in C++ (cont'd.)

- Each operator has a precedence: determines in which order operators in an expression are evaluated
- Operators with lower-precedence numbers are evaluated before higher ones
- Parentheses have lowest-precedence number, so they can be used to override precedence order
- Operators with the same precedence number are evaluated from left to right

Arithmetic Operators in C++ (cont'd.)

Operator	Operation	Precedence number
( )	override normal precedence rules	1
–	negation (reverses the sign of a number)	2
*, /, %	multiplication, division, and modulus arithmetic	3
+, –	addition and subtraction	4

Figure 4-7 Standard arithmetic operators and their order of precedence

Arithmetic Operators in C++ (cont'd.)

Original expression	`13 + 8 / 4 - 2 * 6`
The division is performed first	`13 + 2 - 2 * 6`
The multiplication is performed next	`13 + 2 - 12`
The addition is performed next	`15 - 12`
The subtraction is performed last	`3`
Original expression	`13 + 8 / (4 - 2) * 6`
The subtraction is performed first	`13 + 8 / 2 * 6`
The division is performed next	`13 + 4 * 6`
The multiplication is performed next	`13 + 24`
The addition is performed last	`37`

Figure 4-8 Expressions containing more than one operator having the same precedence

Type Conversions in Arithmetic Expressions

- Recall that the compiler will implicitly promote or demote data types to match when possible
- Sometimes it is necessary to explicitly cast from one data type into another
  - Example: dividing two integers gives the result of integer division (no remainder), but you would really like a `double` result
  - If one or both of the integers is a literal, you can cast it to a `double` by adding `.0` to the end of it
  - If both are variables, you must use the `static_cast` operator

Type Conversions in Arithmetic Expressions (cont'd.)

Example 1 `75.5 * 2`

The integer 2 is implicitly promoted to the **double** number 2.0 before being multiplied by the **double** number 75.5. The result is the **double** number 151.0.

Example 2 `3 * (1.5 + num)`

1. The value stored in the **num** variable (the integer 10) is implicitly promoted to the **double** number 10.0 before it is added to the **double** number 1.5. The result is the **double** number 11.5.
2. The integer 3 is implicitly promoted to the **double** number 3.0 before being multiplied by the **double** number 11.5 (the result of Step 1). The result is the **double** number 34.5.

Example 3 `15 / 0.4`

The integer 15 is implicitly promoted to the **double** number 15.0 before it is divided by the **double** number 0.4. The result is the **double** number 37.5.

Example 4 `num / 4.0`

The value stored in the **num** variable (the integer 10) is implicitly promoted to the **double** number 10.0 before being divided by the **double** number 4.0. The result is the **double** number 2.5.

Figure 4-9 Examples of expressions that require implicit type conversions

The `static_cast` Operator

- Used to explicitly convert data from one data type to another
- Called an **explicit type conversion** or **type cast**
- Syntax: `static_cast<`*dataType*`>(`*data*`)`
  - *data* can be a literal constant, named constant, or variable
  - *dataType* is the data type to which you want the *data* converted

The `static_cast` Operator (cont'd.)

HOW TO Use the static_cast Operator

Syntax

```
static_cast<dataType>(data)
```

Example 1 `static_cast<double>(numA) / static_cast<double>(numB)`

1. The **numA** integer (18) is *explicitly* promoted to the **double** number 18.0.
2. The **numB** integer (4) is *explicitly* promoted to the **double** number 4.0.
3. Step 1's result (18.0) is divided by Step 2's result (4.0), giving the **double** number 4.5.

Example 2 `static_cast<double>(numA) / numB`

1. The **numA** integer (18) is *explicitly* promoted to the **double** number 18.0.
2. The **numB** integer (4) is *implicitly* promoted to the **double** number 4.0.
3. Step 1's result (18.0) is divided by Step 2's result (4.0), giving the **double** number 4.5.

Figure 4-10 How to use the `static_cast` operator

The `static_cast` Operator (cont'd.)

Example 3 `numA / static_cast<double>(numB)`

1. The **numB** integer (4) is *explicitly* promoted to the **double** number 4.0.
2. The **numA** integer (18) is *implicitly* promoted to the **double** number 18.0.
3. Step 2's result (18.0) is divided by Step 1's result (4.0), giving the **double** number 4.5.

Example 4 `7.35 * static_cast<double>(numA)`

1. The **numA** integer (18) is *explicitly* promoted to the **double** number 18.0.
2. Step 1's result (18.0) is multiplied by the **double** number 7.35, giving the **double** number 132.3.

Note: The **static_cast** operator is not required in Example 4 because the computer will *implicitly* convert the contents of the **numA** variable to the **double** data type before performing the multiplication operation.

Example 5 `const float PRICE = static_cast<float>(35.98);`

The **double** number 35.98 is *explicitly* demoted to the **float** data type before being stored in the **PRICE** named constant.

Figure 4-10 How to use the `static_cast` operator (cont'd.)

Assignment Statements

- You use an **assignment statement** to assign a value to a variable while a program is running
- Syntax: *variableName = expression*
  - The = symbol is the **assignment operator**
    - Tells computer to evaluate expression on right side of assignment operator and store result in variable on left side of the operator
  - *expression* can include one or more literal constants, named constants, variables, or arithmetic operators

Assignment Statements (cont'd.)

- Data type of expression in an assignment statement must match data type of the variable
- If they don't match, compiler will use implicit type casting to get them to match
  - Doesn't always produce correct result
  - Better to explicitly cast to correct data type yourself
- Remember:
  - Declaration statement creates a new variable
  - Assignment statement assigns a new value to an existing variable

Assignment Statements (cont'd.)

HOW TO Write an Assignment Statement

Syntax
variableName = expression;

Example 1
```
int population = 0;
population = 5600;
```
The assignment statement assigns the integer 5600 to the `population` variable.

Example 2
```
int females = 5;
int males = 7;
int total = 0;
total = females + males;
```
The assignment statement assigns the integer 12 to the `total` variable.

Example 3
```
int numA = 18;
int numB = 4;
double quotient = 0.0;
quotient = static_cast<double>(numA) / numB;
```
The assignment statement assigns the `double` number 4.5 to the `quotient` variable.

Example 4
```
char middleInitial = ' ';
middleInitial = 'P';
```
The assignment statement assigns the letter P to the `middleInitial` variable.

Example 5
```
string state = "";
state = "NJ";
```
The assignment statement assigns the string "NJ" to the `state` variable.

Example 6
```
const double TAX_RATE = 0.045;
double sale = 50.0;
double tax = 0.0;
tax = sale * TAX_RATE;
```
The assignment statement assigns the `double` number 2.25 to the `tax` variable.

Figure 4-11 How to write an assignment statement

Assignment Statements (cont'd.)

IPO chart information

Input

sale e
sales tax ate

Processing

sales tax

Output

st

Algorithm:

e te t e sale e a sales tax ate
al late t e sales tax lt
t e sale e t e sales tax ate
al late t e st a
tax t t e sale e
s la t e st

C++ instructions

```
double salePrice = 0.0;
double taxRate = 0.0;

double salesTax = 0.0;

double cost = 0.0;

cout << "Enter the sale price: ";
cin >> salePrice;
cout << "Enter the sales tax rate: ";
cin >> taxRate;
salesTax = salePrice * taxRate;
cost = salePrice + salesTax;

cout << "Cost: $" << cost << endl;
```

Figure 4-12 Calculation statements for the Addison O'Reilly problem

Arithmetic Assignment Operators

- Allow you to abbreviate assignment statements that contain an arithmetic operator
- Statement must be of the form *variableName = variableName arithmeticOperator value*
- Abbreviated as *variableName arithmeticOperator = value*
  - Example: `price = price*1.05;` can be abbreviated as `price *= 1.05;`
- Most common operators are `+=` , `-=` , `*=` , `/=` , and `%=`

Arithmetic Assignment Operators (cont'd)

HOW TO Use an Arithmetic Assignment Operator

Syntax

variableName arithmeticAssignmentOperator value;

Operator	Purpose
+=	addition assignment
-=	subtraction assignment
*=	multiplication assignment
/=	division assignment
%=	modulus assignment

Example 1

Original statement:
```
rate = rate + .05;
```
Abbreviated statement:
```
rate += .05;
```

Example 2

Original statement:
```
price = price - discount;
```
Abbreviated statement:
```
price -= discount;
```

Figure 4-13 How to use an arithmetic assignment operator

Step 5–Desk-Check the Program

- Fifth step is to desk-check the program to make sure instructions were translated correctly
- You should desk-check the program using sample data used to desk-check the algorithm
- Results of both desk-checks should be the same
- First, place names of the declared memory locations in a new desk-check table along with each memory location's initial value
- Next, desk-check remaining C++ instructions in order, recording any changes made to the variables

Step 5–Desk-Check the Program (cont'd.)

Figure 4-14 Algorithm's desk-check table from Chapter 2

Figure 4-15 Variable names and initial values entered in the program's desk-check table

Step 5–Desk-Check the Program (cont'd.)

salePrice	taxRate	salesTax	cost
~~0.0~~	~~0.0~~	~~0.0~~	0.0
2300.0	.0		

Figure 4-16 Input values entered in the program's desk-check table

salePrice	taxRate	sales ax	cost
~~0.0~~	~~0.0~~	~~0.0~~	0.0
2300.0	.05	115.0	

Figure 4-17 Sales tax amount entered in the desk-check table

Step 5–Desk-Check the Program (cont'd.)

salePrice	ta ate	sales a	cost
~~0.0~~	~~0.0~~	~~0.0~~	~~0.0~~
2300.0	.05	115.0	2415.0

Figure 4-18 Cost amount entered in the desk-check table

salePrice	ta ate	sales a	cost
~~0.0~~	~~0.0~~	~~0.0~~	~~0.0~~
~~2300.0~~	~~.05~~	~~115.0~~	~~2415.0~~
~~0.0~~	~~0.0~~	~~0.0~~	~~0.0~~
5200.0	.03	15 .0	535 .0

Figure 4-19 Program's desk-check table showing the results of the second desk-check

Step 6–Evaluate and Modify the Program

- Final step in the problem-solving process
- You evaluate a program by running the program on the computer and entering the sample data used when desk-checking the program
- If evaluation reveals errors (known as **bugs**), they must be fixed
- Process of locating and fixing errors is called **debugging**
- Two types of bugs: syntax errors and logic errors

Step 6–Evaluate and Modify the Program (cont'd.)

- **Syntax errors** result from breaking programming language's rules; cause compiler errors
- **Logic errors** don't cause compiler errors; can be hard to identify
  - Example: entering instructions in the wrong order
- Need a text editor to enter C++ instructions
- Instructions are called **source code** and are saved in source files with extension .cpp
- Need a compiler to translate source code into machine code (also called **object code**)

Step 6–Evaluate and Modify the Program (cont'd.)

- Compiler saves object code in **object files** with extension .obj
- **Linker** combines .obj files with other machine code necessary to run the program and produces an executable file with extension .exe
- An **IDE** (**integrated development environment**) is a development tool that contains both an editor and compiler
- A command-line compiler contains only the compiler and requires a separate editor to enter source code

Step 6–Evaluate and Modify the Program (cont'd.)

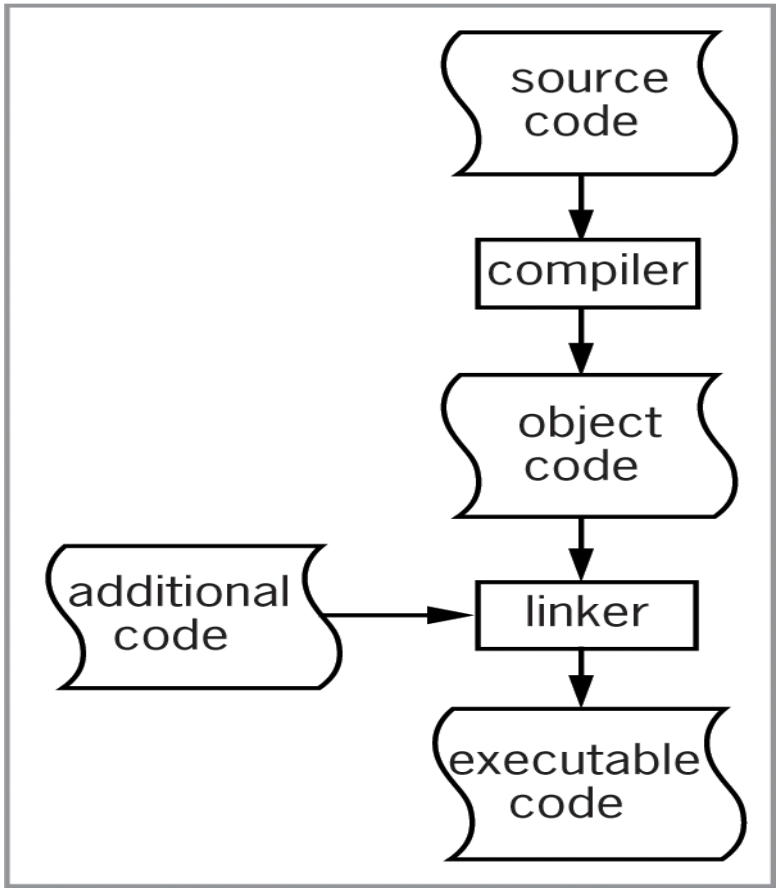

Figure 4-20 Process by which source code is translated into executable code

Step 6–Evaluate and Modify the Program (cont'd.)

- A **comment** is a form of internal documentation; written by placing `//` in front of the comment text
  - Ignored by the compiler
  - Considered good programming practice; makes code more readable
- A **`#include` directive** allows you to merge the source code in one file with that in another file
- The `#include <iostream>` is required when using the `cin` or `cout` stream objects
  - Not a statement, so no semicolon needed at the end

Step 6–Evaluate and Modify the Program (cont'd.)

- A **`using` directive** tells the compiler where in internal memory it can find definitions of C++ keywords and classes like `double` or `string`
- The `using namespace std;` directive indicates that the definitions of the standard C++ keywords and classes are located in the std (standard) namespace
  - Is a statement, so semicolon required at the end
- A namespace is a special area in internal memory

Step 6–Evaluate and Modify the Program (cont'd.)

- A **function** is a block of code that performs a task
- Functions have parentheses following their name (Example: `main()`)
- Some functions require information between the parentheses; others do not
- Every C++ program has one (and only one) `main` function; this is where program execution begins
- Some functions return a value, and the data type they return appears to the left of the function name
  - Example: `int main()`

Step 6–Evaluate and Modify the Program (cont'd.)

- Other functions do not return a value, and `void` appears to the left of the function name
- The return type, name, and parameters (information in parentheses) constitute the **function header**, which marks the beginning of the function
- After the function header, you enter the function's code
- You enclose a function's code in a set of braces (`{ }`)
- The code between the braces is called the **function body**

Step 6–Evaluate and Modify the Program (cont'd.)

```
 1 //Fig4-21.cpp - displays the cost of a TV
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 using namespace std;
 6
 7 int main()                          function header
 8 {
 9    //declare variables
10    double salePrice = 0.0;
11    double taxRate   = 0.0;
12    double salesTax  = 0.0;
13    double cost      = 0.0;
14
15    //enter input items
16    cout << "Enter the sale price: ";
17    cin >> salePrice;
18    cout << "Enter the sales tax rate: ";
19    cin >> taxRate;
20
21    //calculate the sales tax and cost
22    salesTax = salePrice * taxRate;
23    cost = salePrice + salesTax;
24
25    //display output item
26    cout << "Cost: $" << cost << endl;
27
28    return 0;
29 } //end of main function
```

Figure 4-21 Addison O'Reilly program

Step 6–Evaluate and Modify the Program (cont'd.)

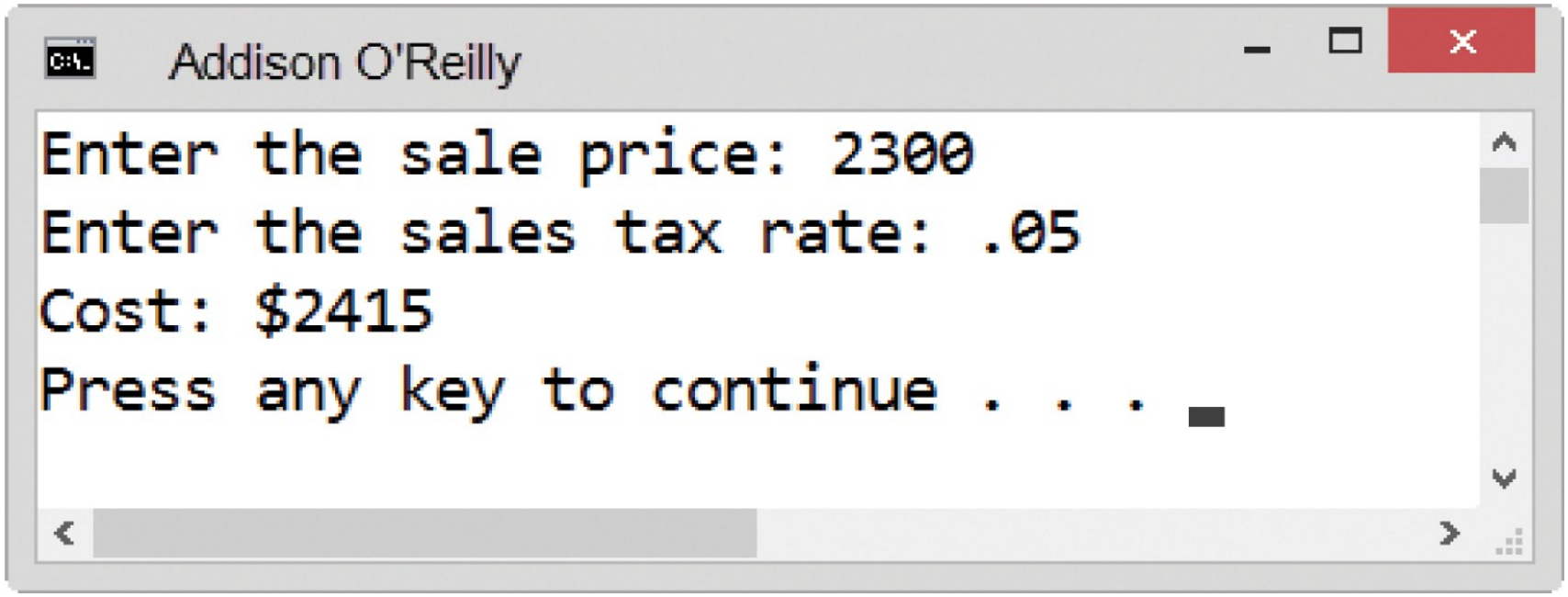

Figure 4-22 Command Prompt window

Summary

- Fourth step in problem-solving process is coding the algorithm into a program
- C++ uses stream objects for standard input/output operations
- Use `cin` with extraction operator (>>) to get numeric or character data from the keyboard
- Use `cout` with insertion operator (<<) to display information on the screen
- The `endl` stream manipulator advances cursor to next line

Summary (cont'd.)

- You do calculations by writing arithmetic expressions using arithmetic operators
- Each operator has a precedence: determines the order of evaluation in an expression
- Parentheses are used to override this order
- Compiler implicitly casts data types when possible, but you should explicitly cast to ensure correctness
- Use the `static_cast` operator to explicitly cast variables from one data type to another

Summary (cont'd.)

- An assignment statement assigns a value to a variable during runtime
- The expression on the right side of the assignment operator (=) in an assignment statement is stored in the variable on its left
- Fifth step of the problem-solving process is to desk-check the program using the same data used to desk-check the algorithm
- The sixth step is to evaluate and modify (if necessary) the program

Summary (cont'd.)

- Errors (called bugs) can either be syntax errors or logic errors
- You need a text editor and compiler to enter C++ instructions and compile them into a program
- C++ instructions are called source code and are saved in source files with the extension .cpp
- The compiler translates source code into machine code, also called object code
- A linker produces an executable file that contains all machine code necessary to run a C++ program

Summary (cont'd.)

- Programmers use comments to document a program internally
  - Comments are not processed by the compiler
- The `#include <`*filename*`>` directive allows you to include multiple source files in a program
- The `using namespace std;` directive tells the compiler where definitions of standard C++ keywords and classes are in internal memory
- A namespace is a special area in the computer's internal memory

Summary (cont'd.)

- Execution of every C++ program begins with the `main()` function
- The first line of a function is the function header
- The function body follows the header and is enclosed in braces
- Some functions return a data type; others return `void`
- Arithmetic assignment operators can be used to abbreviate certain assignment statements with arithmetic operators in them

Lab 4-1: Stop and Analyze

Example 1

```
int quantity = 10;
double itemCost = 5.35;
double amountDue = 0.0;
amountDue = quantity * itemCost + 5;
```

Example 2

```
int total = 30;
total = total / 3;
```

Example 3

```
double store1Sales = 5678.43;
double store2Sales = 8325.72;
double avgSales = 0.0;
avgSales = store1Sales + store2Sales / 2;
```

Example 4

```
int midterm = 74;
int final = 93;
double average = 0.0;
average = (midterm + final) / 2;
```

Figure 4-23 Examples for Lab 4-1

Lab 4-2: Plan and Create

In Chapter 3's Lab 3-2, you planned, created, and desk-checked an algorithm that displays 10% commission on a sales amount. Figure 4-24 shows the algorithm, along with the input and output items and their corresponding C++ statements. It also includes the desk-check table you completed in Lab 3-2.

Figure 4-25 shows the completed IPO chart and C++ instructions for Lab 4-2.

Figure 4-27 shows the completed commission program.

Lab 4-2: Plan and Create (cont'd.)

```
 1 //Lab4-2.cpp - displays a salesperson's commission
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 using namespace std;
 6
 7 int main()
 8 {
 9    //declare named constant and variables
10    const double COMM_RATE = 0.1;
11    double sales           = 0.0;
12    double commission      = 0.0;
13
14    //enter input item
15    cout << "Sales amount: ";
16    cin >> sales;
17
18    //calculate and display the commission
19    commission = sales * COMM_RATE;
20    cout << "Commission: $"
21         <<    commission << endl;
22
23    return 0;
24 } //end of main function
```

Figure 4-27 Commission program

Lab 4-3: Modify

- Modify the program in Lab 4-2 to allow the user to enter the commission rate (in decimal form).
- Test the program twice. For the first test, use 1328.50 and .1 as the sales amount and commission rate. For the second test, use 267.90 and .15.

Lab 4-4: What's Missing?

The program in this lab should calculate and display the volume of a cylinder, given the cylinder's radius (r) and height (h), and using 3.14 as the value of pi.

Lab 4-5: Desk-Check

Desk-check the seven lines of code shown in Figure 4.28 below

```
int num 75;
int result1 = 0;
int result2 = 0;
double result3 = 0.0;
result1 = num % 2;
result2 = num / 2;
result3 = num / 2.0;
```

Lab 4-6: Debug

Following the instructions for Lab 4-6. The program should calculate and display the area of a triangle, but it is not working properly. Run and debug the program.