Introduction to Programming in C++
Seventh 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

Treyson Mobley wants a program that calculates and displays the amount he should tip a waiter at a restaurant. The program should subtract any liquor charge from the total bill and then calculate the tip (using a percentage) on the remainder.

IPO chart information	C++ instructions
Input	
total bill	`double totalBill = 0.0;`
liquor charge	`double liquor = 0.0;`
tip percentage	`double tipPercent = 0.0;`
Processing	
total bill without liquor charge	`double totalNoLiquor = 0.0;`
Output	
tip	`double tip = 0.0;`

Algorithm

1. enter the total bill, liquor charge, and tip percentage
2. calculate the total bill without liquor charge by subtracting the liquor charge from the total bill
3. calculate the tip by multiplying the total bill without liquor charge by the tip percentage
4. display the tip

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.)

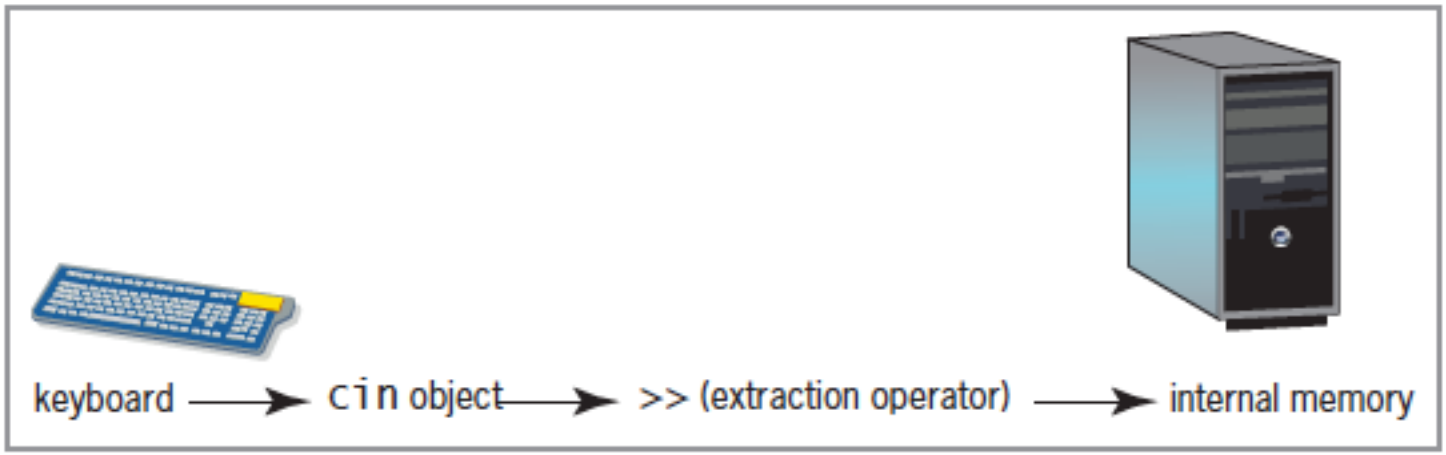


Figure 4-2 Relationship among the keyboard, `cin` object, extraction operator, and internal memory

Getting Data from the Keyboard (cont'd.)

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

Syntax

cin >> *variableName*; — semicolon

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	C++ instructions
Input	
total bill	`double totalBill = 0.0;`
liquor charge	`double liquor = 0.0;`
tip percentage	`double tipPercent = 0.0;`
Processing	
total bill without liquor charge	`double totalNoLiquor = 0.0;`
Output	
tip	`double tip = 0.0;`
Algorithm	

1. enter the total bill, liquor charge, and tip percentage	`cin >> totalBill;`
	`cin >> liquor;`
	`cin >> tipPercent;`

2. calculate the total bill without liquor charge by subtracting the liquor charge from the total bill	
3. calculate the tip by multiplying the total bill without liquor charge by the tip percentage	
4. display the tip	

Figure 4-4 Input statements for the Treyson Mobley 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

you can use more than one insertion operator

Syntax

cout << *item1* [<< *item2* << *itemN*]; — semicolon

Examples

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

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

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

IPO chart information	C++ instructions
Input	
total bill	`double totalBill = 0.0;`
liquor charge	`double liquor = 0.0;`
tip percentage	`double tipPercent = 0.0;`
Processing	
total bill without liquor charge	`double totalNoLiquor = 0.0;`
Output	
tip	`double tip = 0.0;`
Algorithm	

1. enter the total bill, liquor charge, and tip percentage	`cout << "Enter the total bill: ";`
	`cin >> totalBill;`
	`cout << "Enter the liquor charge: ";`
	`cin >> liquor;`
	`cout << "Enter the tip percentage in decimal format: ";`
	`cin >> tipPercent;`

2. calculate the total bill without liquor charge by subtracting the liquor charge from the total bill	
3. calculate the tip by multiplying the total bill without liquor charge by the tip percentage	
4. display the tip	`cout << "Tip: $" << tip << endl;`

Figure 4-6 Prompts and output statement for the Treyson Mobley 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	5 + 12 / 3 – 1
The division is performed first	5 + 4 – 1
The addition is performed next	9 – 1
The subtraction is performed last	8
Original expression	5 + 12 / (3 – 1)
The subtraction is performed first	5 + 12 / 2
The division is performed next	5 + 6
The addition is performed last	11

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 `3 * 1.15`
The integer 3 is implicitly promoted to the `double` number 3.0 before being multiplied by the `double` number 1.15. The result is the `double` number 3.45.

Example 2 `9 * (2.5 + firstNum)`
1. The value stored in the `firstNum` variable (the integer 5) is implicitly promoted to the `double` number 5.0 before it is added to the `double` number 2.5. The result is the `double` number 7.5.

2. The integer 9 is implicitly promoted to the `double` number 9.0 before being multiplied by the `double` number 7.5 (the result of Step 1). The result is the `double` number 67.5.

Example 3 `9.8 / 2`
The integer 2 is implicitly promoted to the `double` number 2.0 before it is divided into the `double` number 9.8. The result is the `double` number 4.9.

Example 4 `firstNum / 2.0`
The value stored in the `firstNum` variable (the integer 5) is implicitly promoted to the `double` number 5.0 before being divided by the `double` number 2.0. 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>(firstNum) / static_cast<double>(secondNum)`

1. The value stored in the `firstNum` variable (the integer 5) is explicitly promoted to the `double` number 5.0.
2. The value stored in the `secondNum` variable (the integer 2) is explicitly promoted to the `double` number 2.0.
3. The `double` number 5.0 (the result of Step 1) is divided by the `double` number 2.0 (the result of Step 2). The result of the division is the `double` number 2.5.

Example 2 `static_cast<double>(firstNum) / secondNum`

1. The value stored in the `firstNum` variable (the integer 5) is explicitly promoted to the `double` number 5.0.
2. The value stored in the `secondNum` variable (the integer 2) is implicitly promoted to the `double` number 2.0.
3. The `double` number 5.0 (the result of Step 1) is divided by the `double` number 2.0 (the result of Step 2). The result of the division is the `double` number 2.5.

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

The `static_cast` Operator (cont'd.)

Example 3 `firstNum / static_cast<double>(secondNum)`

1. The value stored in the `secondNum` variable (the integer 2) is explicitly promoted to the `double` number 2.0.
2. The value stored in the `firstNum` variable (the integer 5) is implicitly promoted to the `double` number 5.0.
3. The `double` number 5.0 (the result of Step 2) is divided by the `double` number 2.0 (the result of Step 1). The result of the division is the `double` number 2.5.

Example 4 `10.65 * static_cast<double>(firstNum)`

The value stored in the `firstNum` variable (the integer 5) is explicitly promoted to the `double` number 5.0 before being multiplied by the `double` number 10.65. The result is the `double` number 53.25. The `static_cast` operator is not required in this example, because the computer will implicitly convert the contents of the `firstNum` variable to the `double` data type before performing the multiplication operation.

Example 5 `const float PRICE = static_cast<float>(3.99);`

The `double` number 3.99 is explicitly converted 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 quantity = 0;
quantity = 1000;
```

The assignment statement assigns the integer 1000 to the `quantity` variable.

Example 2

```
int janOrder = 500;
int febOrder = 225;
int total = 0;
total = janOrder + febOrder;
```

The assignment statement assigns the integer 725 to the `total` variable.

Figure 4-11 How to write an assignment statement

Assignment Statements (cont'd.)

Example 3

```
int firstNum = 5;
int secondNum = 2;
double quotient = 0.0;
quotient = static_cast<double>(firstNum) / secondNum;
```

The assignment statement assigns the double number 2.5 to the quotient variable.

Example 4

```
char middleInitial = ' ';
middleInitial = 'C';
```

The assignment statement assigns the letter C to the middleInitial variable.

Example 5

```
string customerName = "";
customerName = "Jeff Brown";
```

The assignment statement assigns the string "Jeff Brown" to the customerName variable.

Figure 4-11 How to write an assignment statement (cont'd.)

Assignment Statements (cont'd.)

IPO chart information	C++ instructions
Input	
total bill	`double totalBill = 0.0;`
liquor charge	`double liquor = 0.0;`
tip percentage	`double tipPercent = 0.0;`
Processing	
total bill without liquor charge	`double totalNoLiquor = 0.0;`
Output	
tip	`double tip = 0.0;`
Algorithm	

1. enter the total bill, liquor charge, and tip percentage	`cout << "Enter the total bill: ";`
	`cin >> totalBill;`
	`cout << "Enter the liquor charge: ";`
	`cin >> liquor;`
	`cout << "Enter the tip percentage in decimal format: ";`
	`cin >> tipPercent;`

2. calculate the total bill without liquor charge by subtracting the liquor charge from the total bill	`totalNoLiquor = totalBill - liquor;`
3. calculate the tip by multiplying the total bill without liquor charge by the tip percentage	`tip = totalNoLiquor * tipPercent;`
4. display the tip	`cout << "Tip: $" << tip << endl;`

Figure 4-12 Calculation statements for the Treyson Mobley 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.)

total bill	liquor charge	tip percentage	total bill without liquor charge	tip
~~45~~	~~10~~	~~.2~~	~~35~~	~~7~~
30	0	.15	30	4.50

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

totalBill	liquor	tipPercent	totalNoLiquor	tip
0.0	0.0	0.0	0.0	0.0

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

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

totalBill	liquor	tipPercent	totalNoLiquor	tip
~~0.0~~	~~0.0~~	~~0.0~~	0.0	0.0
45.0	10.0	.2		

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

totalBill	liquor	tipPercent	totalNoLiquor	tip
~~0.0~~	~~0.0~~	~~0.0~~	~~0.0~~	0.0
45.0	10.0	.2	35.0	

Figure 4-17 Desk-check table showing the result of the total bill without liquor charge calculation

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

totalBill	liquor	tipPercent	totalNoLiquor	tip
~~0.0~~	~~0.0~~	~~0.0~~	~~0.0~~	~~0.0~~
45.0	10.0	.2	35.0	7.0

Figure 4-18 Desk-check table showing the result of the tip calculation

totalBill	liquor	tipPercent	totalNoLiquor	tip
~~0.0~~	~~0.0~~	~~0.0~~	~~0.0~~	~~0.0~~
~~45.0~~	~~10.0~~	~~.2~~	~~35.0~~	~~7.0~~
~~0.0~~	~~0.0~~	~~0.0~~	~~0.0~~	~~0.0~~
30.0	0.0	.15	30.0	4.50

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.)

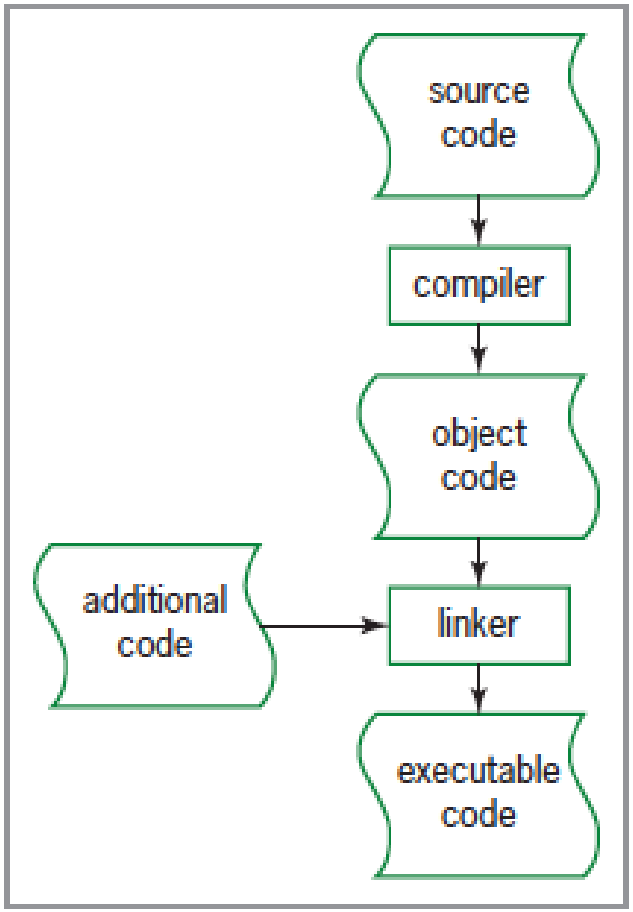


Figure 4-20 Process by which source code is translated into executable code

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

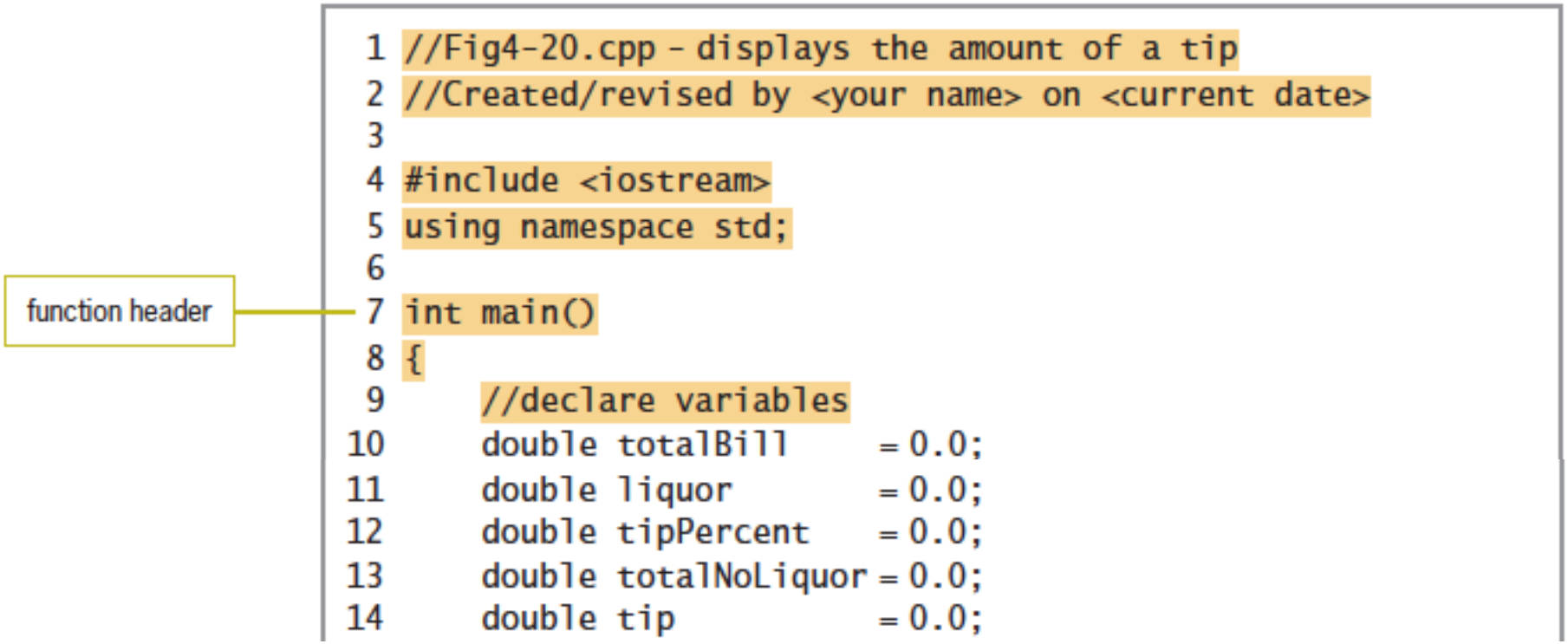
Step 6–Evaluate and Modify the Program (cont'd.)

- 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 //Fig4-20.cpp - displays the amount of a tip
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 using namespace std;
 6
 7 int main()
 8 {
 9     //declare variables
10     double totalBill     = 0.0;
11     double liquor        = 0.0;
12     double tipPercent    = 0.0;
13     double totalNoLiquor = 0.0;
14     double tip           = 0.0;
```

Figure 4-21 Treyson Mobley program

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

```
16      //enter input items
17      cout << "Enter the total bill: ";
18      cin >> totalBill;
19      cout << "Enter the liquor charge: ";
20      cin >> liquor;
21      cout << "Enter the tip percentage in decimal format: ";
22      cin >> tipPercent;
23
24      //calculate the total without liquor and the tip
25      totalNoLiquor = totalBill - liquor;
26      tip = totalNoLiquor * tipPercent;
27
28      //display the output item
29      cout << "Tip: $" << tip << endl;
30
31      system("pause");
32      return 0;
33  }   //end of main function
```

depending on your C++ development tool, this statement may not be necessary

Figure 4-21 Treyson Mobley program (cont'd.)

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

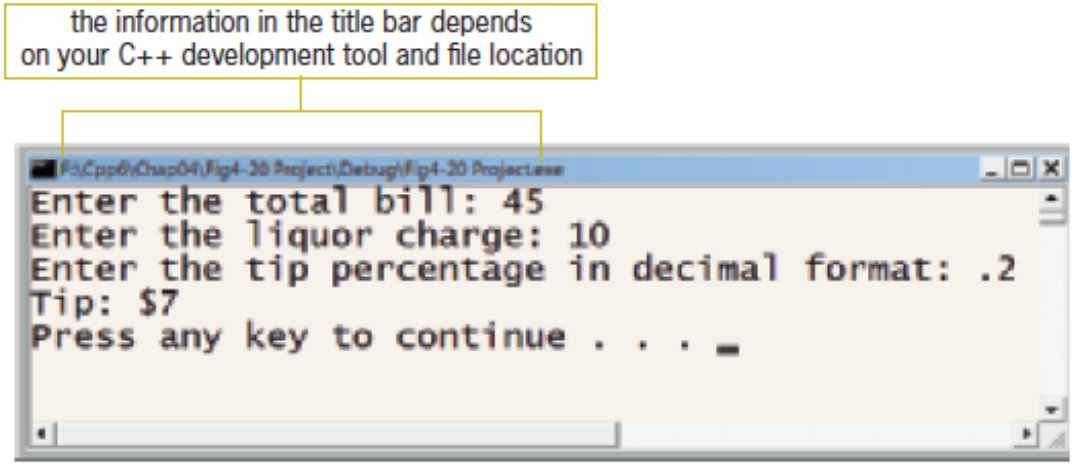


Figure 4-22 Command Prompt window

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 numberOfPeople = 10;
double costPerPerson = 7.45;
double totalCost = 0.0;
totalCost = numberOfPeople * costPerPerson;
numberOfPeople = numberOfPeople / 2;
costPerPerson = costPerPerson + 3;
```

Example 2

```
double score1 = 100.0;
double score2 = 90.0;
double average = 0.0;
average = score1 + score2 / 2;
```

Example 3

```
int juneSales = 933;
int julySales = 1216;
double avgSales = 0.0;
avgSales = (juneSales + julySales) / 2;
```

Figure 4-23 Examples for Lab 4-1

Lab 4-2: Plan and Create

- Plan and create an algorithm that displays the total amount a student owes for a semester

> The cashier at Hoover College wants a program that calculates and displays the total amount a student owes for the semester, including tuition and room and board. The fee per semester hour is $100, and room and board is $2000 per semester. Courses at the college can be 1, 2, or 3 semester hours.

Figure 4-24 Problem specification for Lab 4-2

Lab 4-3: Modify

- Modify the program in Lab 4-2 to account for Hoover College having courses that can be 0.5, 1, 2, or 3 semester hours
- Additionally, fee per hour is raised to $105
- Test the program twice, using 9.5 and 11 as the number of hours enrolled

Lab 4-4: Desk-Check

- Desk-check the three lines of code shown in Figure 4-31

```
int num 75;
int answer = 0;
answer = num % 2;
```

Figure 4-31 Code for Lab 4-4

Lab 4-5: Debug

- Follow the instructions for starting C++ and opening the Lab4-5.cpp file
- If necessary, make the `system("pause");` statement a comment, and then save the program
- Run and then debug the program