Introduction to Programming in C++
Eighth Edition

Chapter 5:
The Selection Structure

Objectives

- Include the selection structure in pseudocode and in a flowchart
- Code a selection structure using the `if` statement
- Include comparison operators in a selection structure's condition
- Include logical operators in a selection structure's condition
- Temporarily convert a character to either uppercase or lowercase
- Format numeric output

Making Decisions

- With only the sequence structure, all instructions are executed in order
- A **selection structure** is needed when a decision must be made (based on some condition) before selecting an instruction to execute
- A selection structure's condition must be phrased as a Boolean expression (evaluates to true or false)

Making Decisions (cont'd.)

- **Single-alternative selection structure**
  - Set of instructions is executed only if condition evaluates to true
- **Dual-alternative selection structure**
  - Executes different sets of instructions based on whether condition evaluates to true or false
- **True path**
  - Instructions followed when condition evaluates to true
- **False path**
  - Instructions followed when condition evaluates to false

Making Decisions (cont'd.)

Problem specification

Dr. N is sitting in a chair in his lair, facing a control deck and an electronic screen. At times, visitors come to the door located at the rear of the lair. Before opening the door, which is accomplished by pressing the blue button on the control deck, Dr. N likes to view the visitor on the screen; he can do this by pressing the orange button on the control deck. Write the instructions that direct Dr. N to view the visitor first, and then open the door and say "Welcome".

1. press the orange button on the control deck to view the visitor on the screen
2. press the blue button on the control deck to open the door
3. say "Welcome"

Figure 5-1 A problem that requires the sequence structure only

Making Decisions (cont'd.)

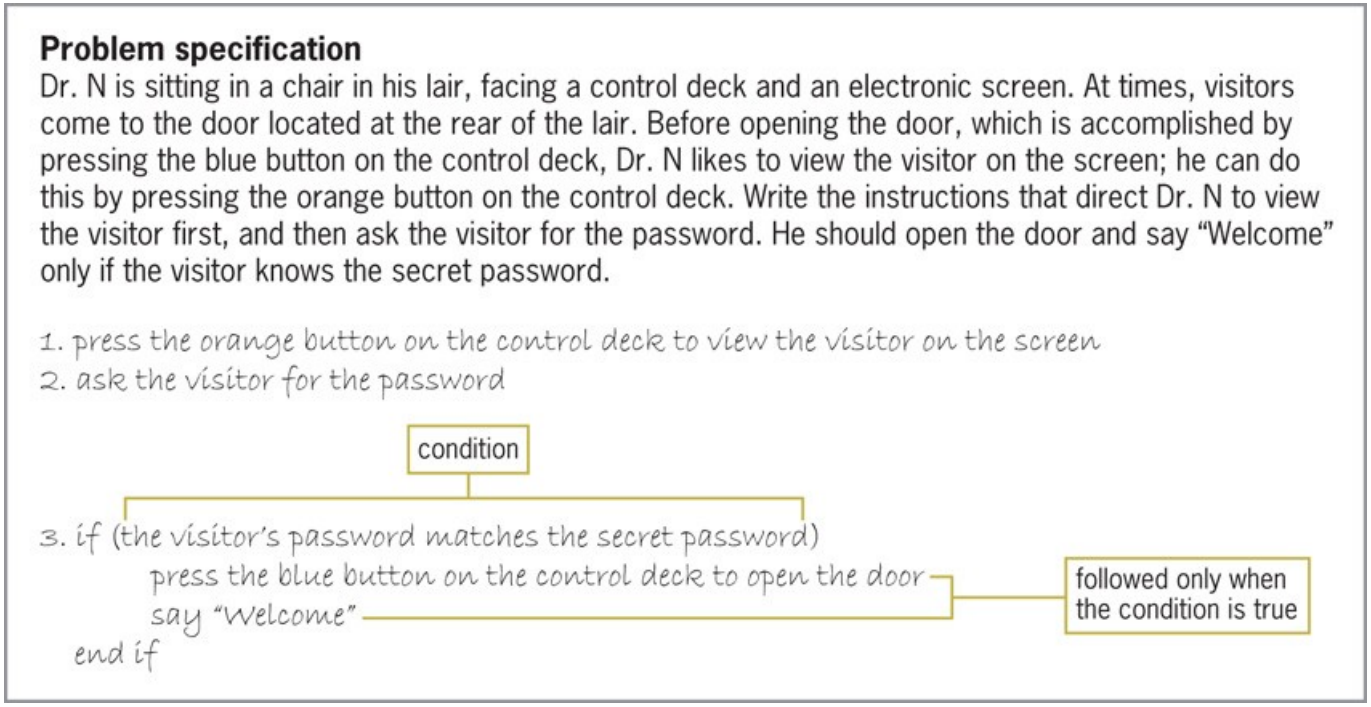

Figure 5-2 A problem that requires the sequence structure and a single-alternative selection structure

Making Decisions (cont'd.)

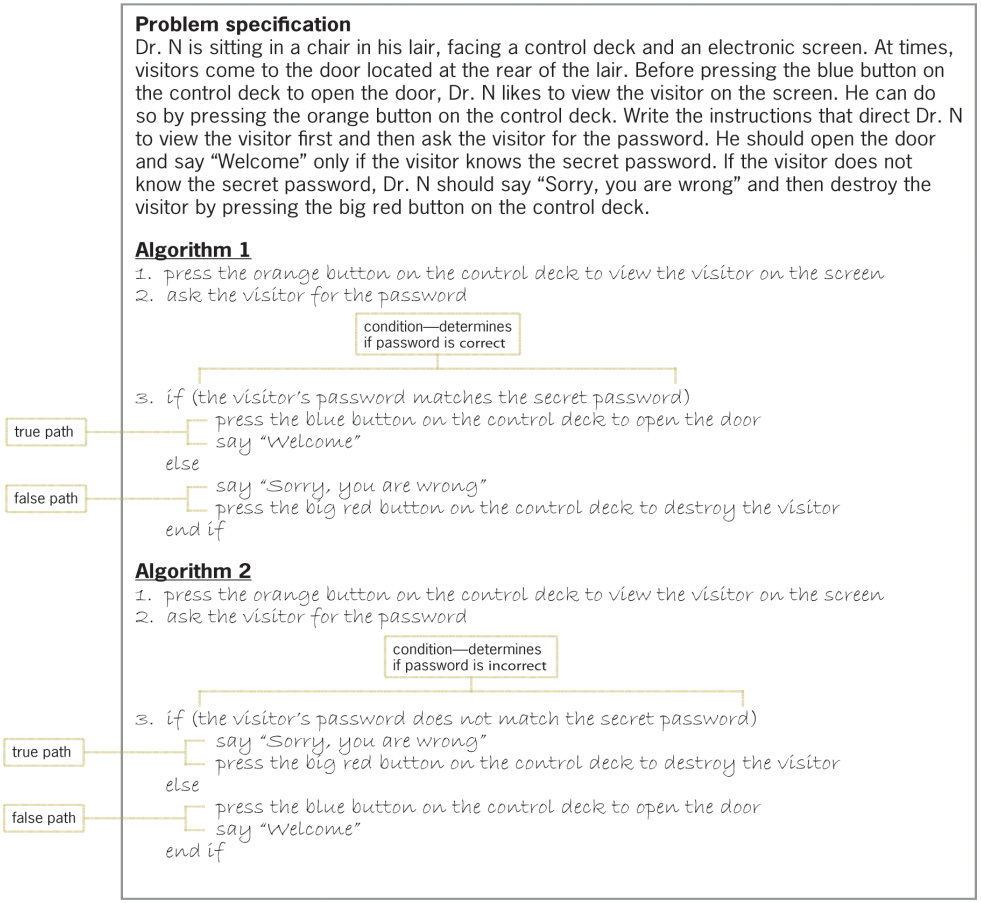

Figure 5-3 A problem that requires the sequence structure and a dual-alternative selection structure

Flowcharting a Selection Structure

- Recall from Chapter 2:
  - Oval = start/stop symbol; rectangle = process symbol; parallelogram = input/output symbol
- Diamond symbol is called **decision symbol**
  - Used to represent condition (decision) in selection and repetition structures
- Selection structures have one flowline leading in and two leading out
  - “T” line leading out points to true path
  - “F” line leading out points to false path

Flowcharting a Selection Structure (cont'd)

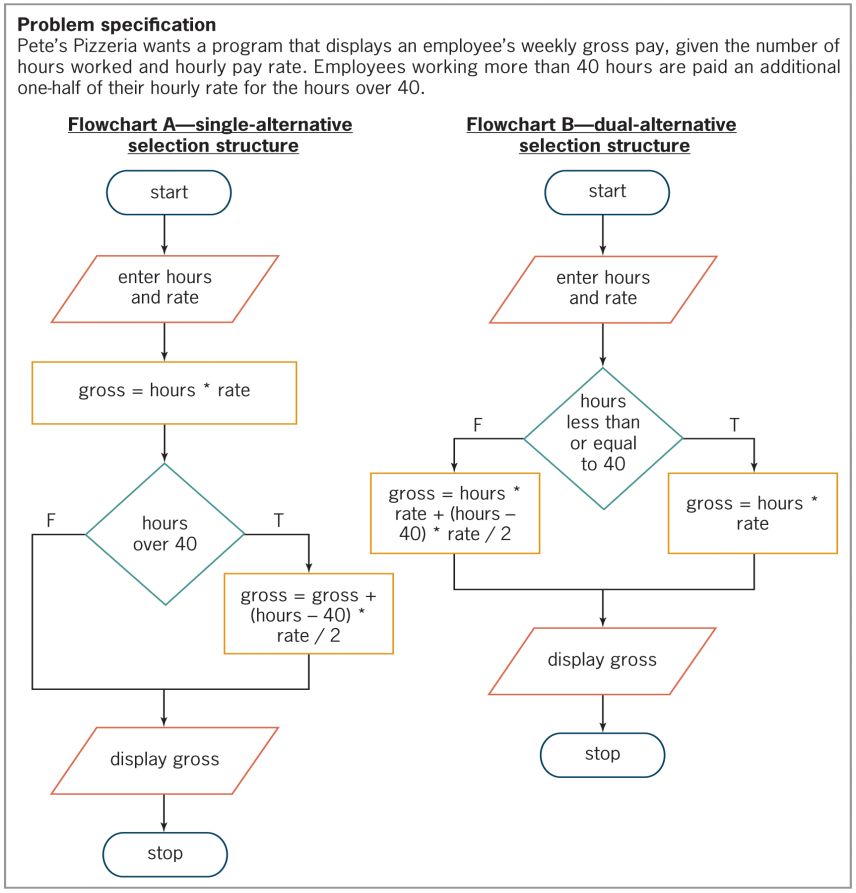

Figure 5-4 Problem specification and two correct algorithms in flowchart form

Coding a Selection Structure in C++

- The `if` (and `else`) statement is used to code most selection structures in C++
- Syntax

  `if` (*condition*)

  one or more statements (true path)

  [`else`

  one or more statements (false path)]
- Keyword `if` and *condition* are required
- Portion in brackets (`else` clause) is optional
  - Only used for dual-alternative selection structures

Coding a Selection Structure in C++ (cont'd.)

- Programmer must supply *condition* and statements in true (and, optionally, false) path
- If path contains more than one statement, must be entered as a **statement block** (enclosed in `{}`)
- Good programming practice to put comment at end of clause (example: `//end if`)
  - Makes program easier to read and understand
- The condition must be a Boolean expression
  - May contain variables, constants, arithmetic operators, comparison operators, and logical operators

Coding a Selection Structure in C++ (cont'd.)

HOW TO Use the `if` Statement

Syntax

```
if (condition)
      one or more statements to be processed when the condition is true
[else
      one or more statements to be processed when the condition is false]
//end if
```

Example 1—one statement in only the true path

```
if (condition)
      one statement
//end if
```

Example 2—multiple statements in only the true path

```
if (condition)
{
      multiple statements enclosed in braces
}     //end if
```

(continues)

Figure 5-5 How to use the `if` statement

Coding a Selection Structure in C++ (cont'd.)

(continued)

Example 3—one statement in each path

```
if (condition)
      one statement
else
      one statement
//end if
```

Example 4—multiple statements in the true path and one statement in the false path

```
if (condition)
{
      multiple statements enclosed in braces
}
else
      one statement
//end if
```

Example 5—one statement in the true path and multiple statements in the false path

```
if (condition)
      one statement
else
{
      multiple statements enclosed in braces
} //end if
```

Example 6—multiple statements in both paths

```
if (condition)
{
      multiple statements enclosed in braces
}
else
{
      multiple statements enclosed in braces
} //end if
```

Figure 5-5 How to use the `if` statement (cont'd)

Comparison Operators

- **Comparison operators** are used to compare two values that have the same data type
  - *less than* (<)
  - *less than or equal to* (<=)
  - *greater than* (>)
  - *greater than or equal to* (>=)
  - *equal to* (==)
  - *not equal to* (!=)
- No spaces between dual-character symbols

Comparison Operators (cont'd.)

- Have precedence numbers like arithmetic operators
- <, <=, >, and >= have precedence value 1
- == and != have precedence value 2
- Lower precedence evaluated first
- Equal precedence evaluated left to right
- Parentheses used to override precedence order

Comparison Operators (cont'd.)

- Expressions with comparison operators always evaluate to Boolean values
- Don't confuse equality operator (==) with assignment operator (=)
- Don't use equality (==) or inequality operator (! =) to compare real numbers
  - Real numbers cannot be stored exactly
  - Instead, test that absolute value of their difference is within some small threshold

Comparison Operators (cont'd.)

HOW TO Use Comparison Operators in an if Statement's Condition

Operator	Operation	Precedence number
<	less than	1
<=	less than or equal to	1
>	greater than	1
>=	greater than or equal to	1
==	equal to	2
!=	not equal to	2

Examples (All of the variables have the `int` data type.)

```
if (quantity < 50)
if (age >= 25)
if (onhand == target)
if (quantity != 7500)
```

Note: When making comparisons, keep in mind that equal to (==) is the opposite of not equal to (!=), greater than (>) is the opposite of less than or equal to (<=), and less than (<) is the opposite of greater than or equal to (>=).

Figure 5-6 How to use comparison operators in an `if` statement's condition

Comparison Operators (cont'd.)

Original expression	`7 - 3 + 8 < 9 + 5`
The subtraction is performed first	`4 + 8 < 9 + 5`
The first addition is performed next	`12 < 9 + 5`
The second addition is performed next	`12 < 14`
The < comparison is performed last	`true`

Figure 5-7 Evaluation steps for an expression containing arithmetic and comparison operators

Swapping Numeric Values

- Example program (next slide) takes in two integers, swaps them if first is greater, and then prints out lower number, followed by higher number
- Uses single-alternative selection structure with statement block in true path
- Creates temporary variable to accomplish swap
- Temp variable can only be used in statement block in which it is declared; called a **local variable**

Swapping Numeric Values (cont'd.)

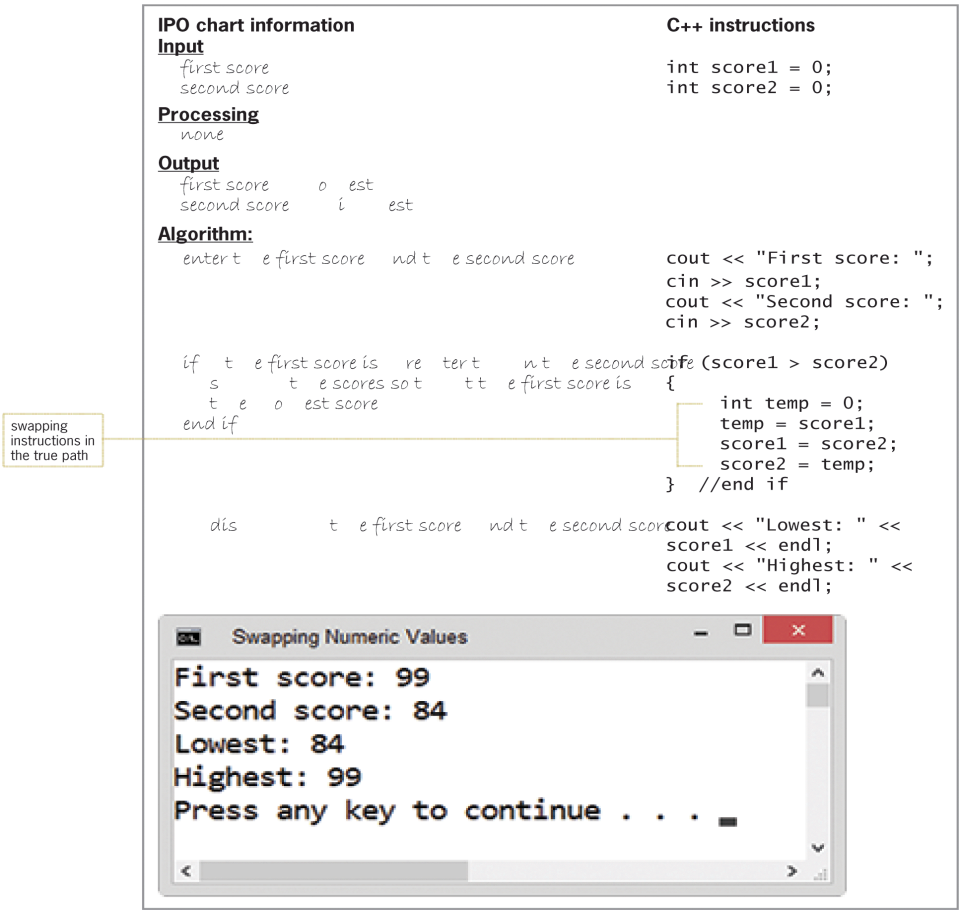

Figure 5-8 Swapping program

Swapping Numeric Values (cont'd.)

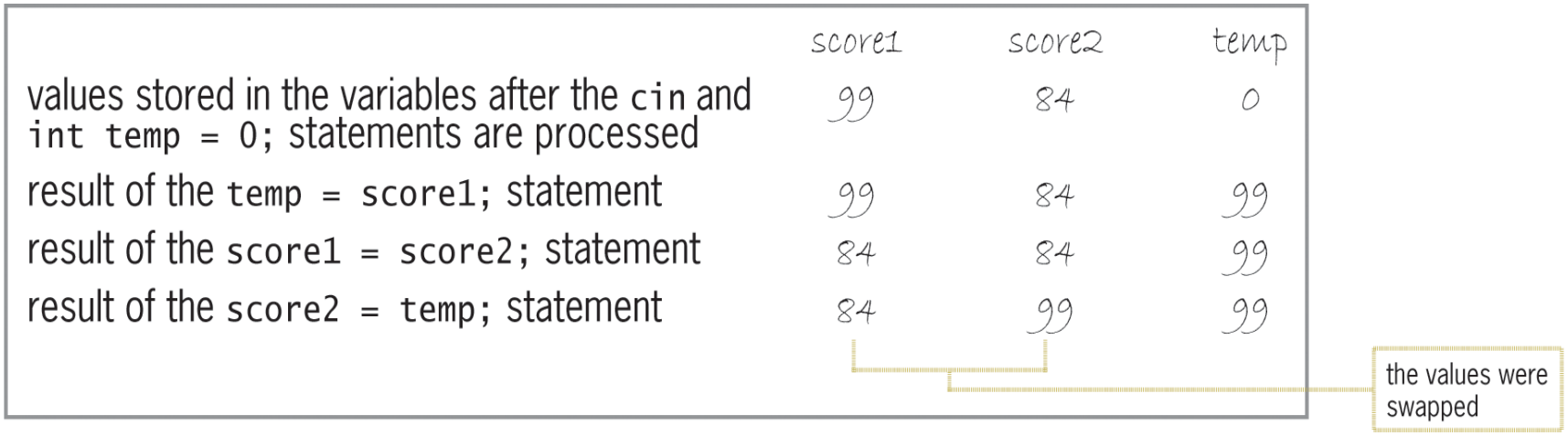

Figure 5-9 Illustration of the swapping concept

Swapping Numeric Values (cont'd.)

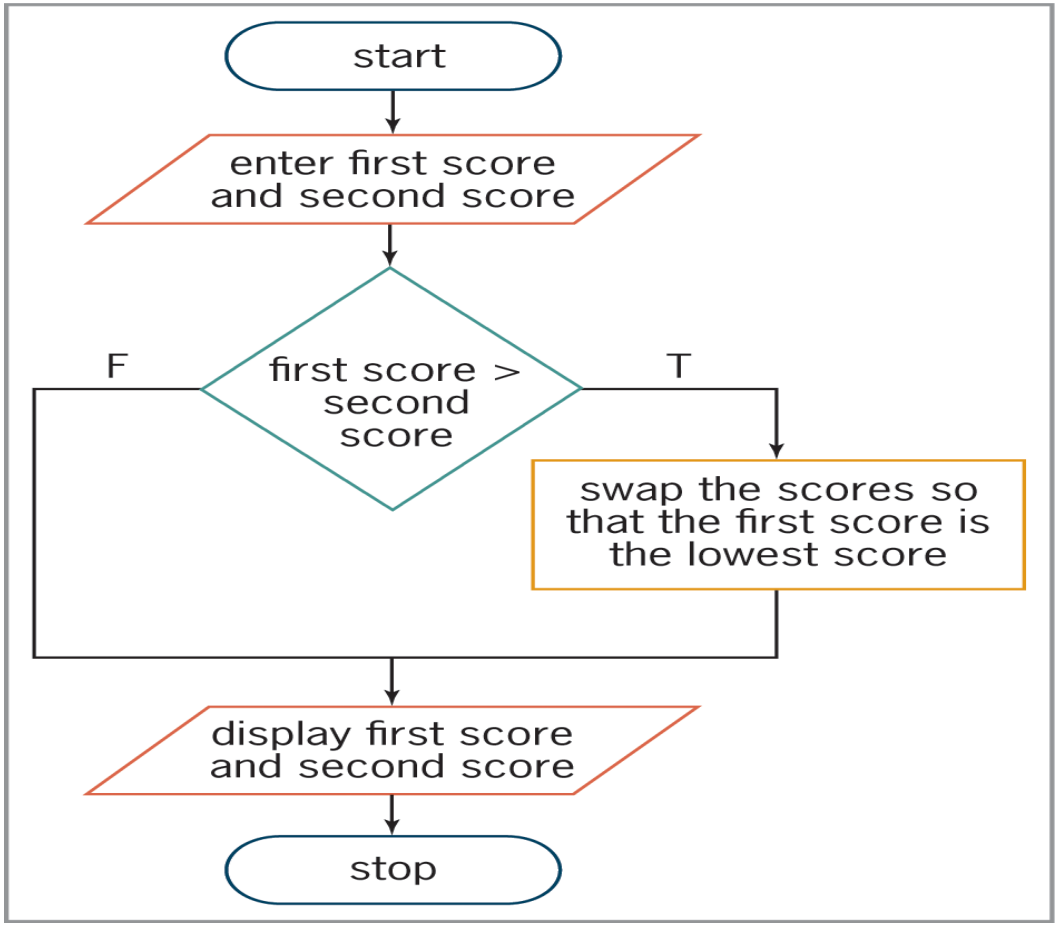

Figure 5-10 Flowchart for the swapping program

Displaying the Area or Circumference

- Example program (next slide) displays the area or circumference of a circle given the radius
- Choice of area or circumference is selected by user
- Uses a dual-alternative selection structure

Displaying the Area or Circumference (cont'd.)

IPO chart information	C++ instructions
Input	
pi (3.14)	`const double PI = 3.14;`
radius	`double radius = 0.0;`
choice	`int choice = 0;`
Processing	
o e	
Output	
ei her he area or he circu ere ce	`double answer = 0.0;`
Algorithm:	

1. e er he radius a d choice	`cout << "Enter the radius: ";`
	`cin >> radius;`
	`cout << "Enter 1 (area) or 2 (circumference): ";`
	`cin >> choice;`

i (he choice is 1)	`if (choice == 1)` `{`
ca cu a e he area u ip i answer = radius * radius * PI; radius i se a d he u ip i he resu pi	
disp a rea a d he area	`cout << "Area: " <<` `answer << endl;`
	`}`
e se	`else` `{`
ca cu a e he circu ere ce u ip i he radius a d he u ip i he resu pi	`answer = 2 * radius * PI;`
disp a ircu ere ce a d he circu ere ce	`cout << "Circumference: " <<` `answer << endl;`
e d i	`} //end if`

Figure 5-11 IPO chart information and C++ instructions for the area or circumference program

Displaying the Area or Circumference (cont'd.)

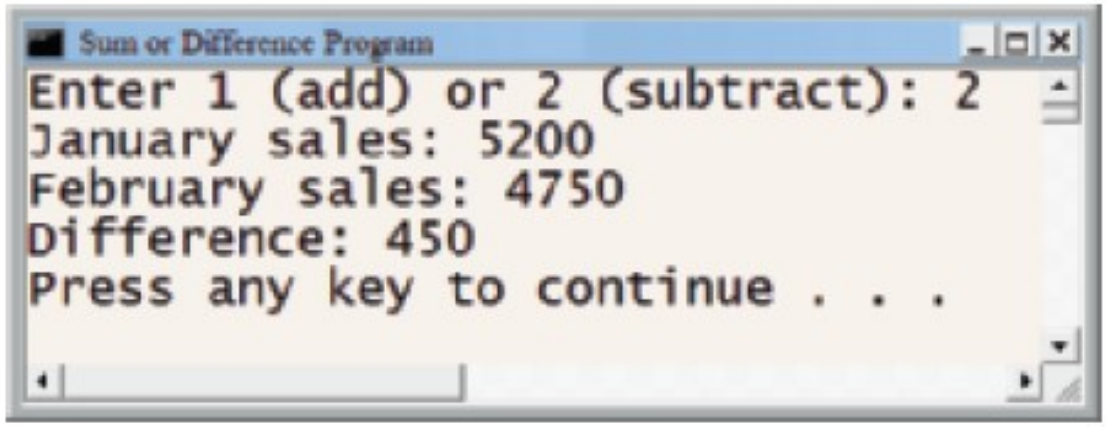

Figure 5-11 Sample run of the area or circumference program

Displaying the Area or Circumference (cont'd.)

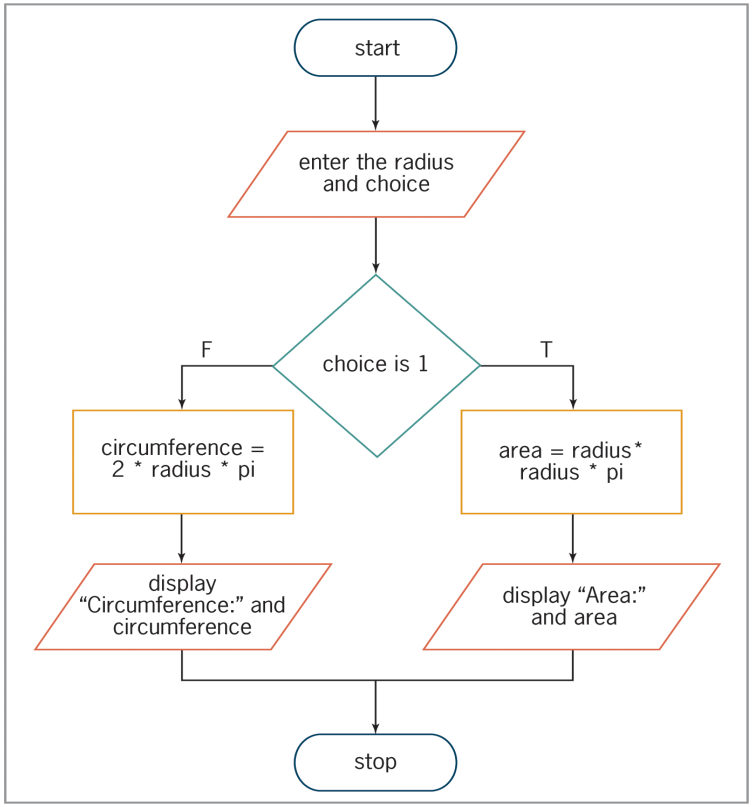

Figure 5-12 Flowchart for area or circumference program

Logical Operators

- **Logical operators** allow you to combine two or more conditions (sub-conditions) into one compound condition
- Also called as **Boolean operators** (always evaluate to true or false)
- Two most common are And (&&) and Or (||)
- All sub-conditions must be true for a compound condition using And to be true
- Only one of the sub-conditions must be true for a compound condition using Or to be true

Logical Operators (cont'd.)

- And evaluated before Or in an expression
- Logical operators evaluated after arithmetic and comparison operators
- Parentheses can override precedence ordering
- **Truth tables** summarize how computer evaluates logical operators
- Only necessary sub-conditions are evaluated; called **short-circuit evaluation**
  - Example: if first sub-condition in an And clause is false, second sub-condition need not be evaluated

Logical Operators (cont'd.)

HOW TO Use Logical Operators in an if Statement's Condition

Operator	Operation	Precedence number
And (&&)	*all* subconditions must be true for the compound condition to evaluate to true	1
Or (||)	only *one* of the subconditions needs to be true for the compound condition to evaluate to true	2

Example 1

```
int population = 0;
cin >> population;
if (population > 2500 && population < 5000)
```

The compound condition evaluates to true when the number stored in the `population` variable is greater than 2500 and, at the same time, less than 5000; otherwise, it evaluates to false.

Example 2

```
int age = 0;
cin >> age;
if (age == 21 || age > 55)
```

The compound condition evaluates to true when the number stored in the `age` variable is either equal to 21 or greater than 55; otherwise, it evaluates to false.

Figure 5-14 How to use logical operators in an `if` statement's condition

Logical Operators (cont'd.)

Example 3

```
int quantity = 0;
double price = 0.0;
cin >> quantity;
cin >> price;
if (quantity < 100 && price <= 10.35)
```

The compound condition evaluates to true when the number stored in the `quantity` variable is less than 100 and, at the same time, the number stored in the `price` variable is less than or equal to 10.35; otherwise, it evaluates to false.

Example 4

```
int quantity = 0;
double price = 0.0;
cin >> quantity;
cin >> price;
if (quantity > 0 && quantity < 100 || price > 34.55)
```

The compound condition evaluates to true when either (or both) of the following is true: the number stored in the `quantity` variable is between 0 and 100 or the number stored in the `price` variable is greater than 34.55; otherwise, it evaluates to false. (The `&&` operator is evaluated before the `||` operator because it has a higher precedence.)

Figure 5-14 How to use logical operators in an `if` statement's condition

Logical Operators (cont'd.)

Truth table for the And (&&) operator

subcondition1	subcondition2	subcondition1 && subcondition2
true	true	true
true	false	false
false	true (not evaluated)	false
false	false (not evaluated)	false

evaluates to true only when both subconditions are true

Truth table for the Or (||) operator

subcondition1	subcondition2	subcondition1 || subcondition2
true	true (not evaluated)	true
true	false (not evaluated)	true
false	true	true
false	false	false

evaluates to false only when both subconditions are false

Figure 5-15 Truth tables for the logical operators

Using the Truth Tables

- Two example problem descriptions are given, and truth tables for And and Or operators are used to find appropriate sub-conditions and logical operators
- Calculate bonus for A-rated salespeople with monthly sales greater than $5000

```
rating == 'A' && sales > 5000
```

- Send letter to all A-rated and B-rated salespeople

```
rating == 'A' || rating == 'B'
```

Calculating Gross Pay

- Example problem description is given in which the gross pay of an employee must be calculated
- Program must verify that number of hours worked is between 0 and 40
- Process of verifying that input data is within expected range is known as **data validation**
- Program outputs gross pay if the number of hours worked is valid and is an error message otherwise

Calculating Gross Pay (cont'd.)

Example 1

```
const int PAY_RATE = 10;
int hoursWorked    = 0;
int grossPay       = 0;

cout  << "Hours worked (0 through 40): ";
cin  >> hoursWorked;

if (hoursWorked >= 0 && hoursWorked <= 40)
{
    grossPay = hoursWorked * PAY_RATE;
    cout << "Gross pay: $" << grossPay << endl;
}
else
    cout << "Incorrect number of hours" << endl;
//end if
```

And operator

Example 2

```
const int PAY_RATE = 10;
int hoursWorked    = 0;
int grossPay       = 0;

cout << "Hours worked (0 through 40): ";
cin >> hoursWorked;

if (hoursWorked < 0 || hoursWorked > 40)
    cout << "Incorrect number of hours" << endl;
else
{
    grossPay = hoursWorked * PAY_RATE;
    cout << "Gross pay: $" << grossPay << endl;
}   //end if
```

Or operator

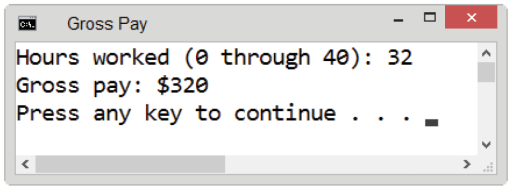

Figure 5-17 Gross Pay program

Different version of Area or Circumference program

Example 1

```
const double PI = 3.14;
double radius = 0.0;
char choice = ' ';
double answer = 0.0;

cout << "Circle Area or Circumference Calculator" << endl;
cout << "Enter the radius: ";
cin >> radius;
cout << "Enter A (area) or C (circumference): ";
cin >> choice;

//calculate and display
if (choice == 'A' || choice == 'a')
{
     answer = radius * radius * PI;
     cout << "Area: " << answer << endl;
}
else
{
     answer = 2 * radius * PI;
     cout << "Circumference: " << answer << endl;
}  //end if
```

character literal constants are enclosed in single quotation marks

Or operator

string literal constants are enclosed in double quotation marks

Example 2

```
const double PI = 3.14;
double radius = 0.0;
char choice = ' ';
double answer = 0.0;

cout << "Circle Area or Circumference Calculator" << endl;
cout << "Enter the radius: ";
cin >> radius;
cout << "Enter A (area) or C (circumference): ";
cin >> choice;

//calculate and display
if (choice != 'A' && choice != 'a')
{
     answer = 2 * radius * PI;
     cout << "Circumference: " << answer << endl;
}
else
{
     answer = radius * radius * PI;
     cout << "Area: " << answer << endl;
}    //end if
```

And operator

```
Area or Circumference
Circle Area or Circumference Calculator
Enter the radius: 15
Enter A (area) or C (circumference): a
Area: 706.5
Press any key to continue . . .
```

Figure 5-18 Different version of Area or Circumference program

Summary of Operators

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
<, <=, >, >=	less than, less than or equal to, greater than, greater than or equal to	5
==, !=	equal to, not equal to	6
And (&&)	*all* subconditions must be true for the compound condition to evaluate to true	7
Or (||)	only *one* of the subconditions needs to be true for the compound condition to evaluate to true	8

Figure 5-19 Listing and an example of arithmetic, comparison, and logical operators

Summary of Operators (cont'd.)

Example	
Original expression	`20 < 80 / 2 + 3 && 25 > 10 * 2`
`80 / 2` is performed first	`20 < 40 + 3 && 25 > 10 * 2`
`10 * 2` is evaluated next	`20 < 40 + 3 && 25 > 20`
`40 + 3` is evaluated next	`20 < 43 && 25 > 20`
`20 < 43` is evaluated next	`true && 25 > 20`
`25 > 20` is evaluated next	`true && true`
`true && true` is evaluated last	`true`

Figure 5-19 Listing and an example of arithmetic, comparison, and logical operators

Converting a Character to Uppercase or Lowercase

- **`toupper`** and **`tolower`** functions used to convert characters between uppercase and lowercase
- `toupper` function temporarily converts a character to uppercase; `tolower` function temporarily converts a character to lowercase
- Syntax is `toupper(`*charVariable*`)` and `tolower(`*charVariable*`)`
- Item between parentheses in a function's syntax is called an **argument** – information the function needs to perform its task

Converting a Character to Uppercase or Lowercase (cont'd.)

- Each function copies the character in its argument to a temporary location in internal memory, converts the character to the appropriate case, and returns the temporary character
- Neither function changes the contents of its argument, but rather, changes the temporary variable

Converting a Character to Uppercase or Lowercase (cont'd.)

HOW TO Use the toupper and tolower Functions

Syntax

toupper(*charVariable*)
tolower(*charVariable*)

Example 1

```
if (toupper(senior) == 'Y')
```

temporarily converts the contents of the `senior` variable to uppercase and then compares the result with the uppercase letter Y

Example 2

```
if (tolower(senior) == 'y')
```

temporarily converts the contents of the `senior` variable to lowercase and then compares the result with the lowercase letter y

Example 3

```
initial = toupper(initial);
senior = tolower(senior);
```

changes the contents of the `initial` and `senior` variables to uppercase and lowercase, respectively

Figure 5-20 How to use the `toupper` and `tolower` functions

Formatting Numeric Output

- Real numbers are displayed in either fixed-point or scientific (e) notation
- Small real numbers (six or less digits before decimal place) displayed in fixed-point notation
  - Example: 1,234.56 displayed as 1234.560000
- Large real numbers (more than six digits before decimal place) displayed in e notation
  - Example: 1,225,000.00 displayed as 1.225e+006
- Purpose of program determines appropriate format

Formatting Numeric Output (cont'd.)

- Stream manipulators allow control over formatting
- **`fixed` stream manipulator** displays real numbers in fixed-point notation
- **`scientific` stream manipulator** displays real numbers in e notation
- Stream manipulator must appear in a `cout` statement before numbers you want formatted
- Manipulator can appear by itself in a `cout` statement or can be included with other information

Formatting Numeric Output (cont'd.)

- Manipulator remains in effect until end of program execution or until another manipulator is processed
- Numbers formatted with `fixed` stream manipulator always have six numbers to the right of the decimal place
  - Number is padded with zeros if there aren't six digits
    - Example: 123.456 is displayed as 123.456000
  - Number is rounded and truncated if there are more than six digits
    - Example: 123.3456789 is displayed as 123.345679

Formatting Numeric Output (cont'd.)

- **`setprecision` stream manipulator** controls number of decimal places
- Syntax `setprecision(`*numberOfDecimalPlaces*`)`
- You can include `setprecision` and `fixed` manipulators in the same `cout` statement
- Definition of `setprecision` manipulator contained in iomanip file
- Program must contain `#include <iomanip>` directive to use `setprecision` manipulator

Formatting Numeric Output (cont'd.)

HOW TO Use the fixed and scientific Stream Manipulators

Example 1

```
double sales = 10575.25;
cout << fixed;
cout << sales << endl;
```

Result: displays 10575.250000

Example 2

```
double rate = 5.9018432;
cout << fixed << rate << endl;
```

Result: displays 5.901843

Example 3

```
double rate = 5.9018436;
cout << fixed << rate << endl;
```

Result: displays 5.901844

Example 4

```
double sales = 10575.25;
cout << scientific << sales << endl;
```

Result: displays 1.057525e+04

Figure 5-21 How to use the `fixed` and `scientific` stream manipulators

Formatting Numeric Output (cont'd.)

HOW TO Use the setprecision Stream Manipulator

Syntax
setprecision(*numberOfDecimalPlaces*)

Example 1

```
double sales = 3500.6;
cout << fixed;
cout << setprecision(2);
cout << sales << endl;
```

Result: displays 3500.60

Example 2

```
double rate = 10.0732;
cout << fixed << setprecision(3);
cout << rate << endl;
```

Result: displays 10.073

Example 3

```
double sales = 3467.55;
cout << fixed;
cout << setprecision(0) << sales;
```

Result: displays 3468

Figure 5-22 How to use the `setprecision` stream manipulator

Summary

- Selection structure used when you want a program to make a decision before choosing next instruction
- Selection structure's condition must evaluate to true or false
- In single-alternative and dual-alternative selection structures, the instructions followed when the condition is true are placed in the true path
- In dual-alternative selection structures, the instructions followed when the condition is false are placed in the false path

Summary (cont'd.)

- A diamond (decision symbol) is used to represent a selection structure's condition in a flowchart
- Each selection structure has one flowline going into the diamond and two coming out ("T" line represents the true path, and "F" line the false path)
- The `if` statement is used to code most selection structures
- True or false paths with more than one statement must be entered as a statement block (enclosed in `{ }`)

Summary (cont'd.)

- Good practice to end `if` and `else` statements with a comment (`//end if`)
- Comparison operators are used to compare values – Expressions using them evaluate to true or false
- Comparison operators have precedence ordering similar to arithmetic operators
- Don't use == and != to compare real numbers
- Local variables can only be used in the statement block in which they are declared

Summary (cont'd.)

- Expressions with logical operators evaluate to true or false
- And (`&&`) and Or (`||`) are logical operators, which also have precedence
- Arithmetic operators are evaluated first in an expression, followed by comparison operators and then logical operators
- Character comparisons are case sensitive
- `toupper` and `tolower` functions temporarily convert a character to upper or lowercase

Summary (cont'd.)

- The `fixed` and `scientific` stream manipulators allow you to format real numbers
- The `setprecision` manipulator allows you to specify the number of decimal places that appear
- `fixed` and `scientific` are defined in the iostream file
- `setprecision` is defined in the iomanip file

Lab 5-1: Stop and Analyze

```
 1 //Lab5-1.cpp - displays projected sales
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <iomanip>
 6 using namespace std;
 7
 8 int main()
 9 {
10     double sales = 0.0;
11     double rate  = 0.0;
12     char code    = ' ';
13
14     cout << "Sales: ";
15     cin >> sales;
16     cout << "Code (1, 2, 3, or 4): ";
17     cin >> code;
18
19     if (code == '1' || code == '3')
20         rate = 0.2;
21     else
22         rate = 0.15;
23     //end if
24
25     //calculate and display the projected sales amount
26     sales = sales + sales * rate;
27     cout << fixed << setprecision(2);
28     cout << "Projected sales: " << sales << endl;
29
30     return 0;
31 }  //end of main function
```

Figure 5-23 Program for Lab 5-1

Lab 5-2: Plan and Create

- Plan and create an algorithm for the Heaton Boutique problem below

Problem specification

Heaton Boutique allows customers to purchase items over the phone and have the items shipped to their homes. The shipping fee is $0.99 if the purchase amount after subtracting any discount is at least $100; otherwise, it is $4.99. The only discount Heaton Boutique offers is to customers who are members of the store's Premier Club; the discount rate is 10%. The program should display the total amount the customer owes for his or her purchase.

Figure 5-24 Problem specification for Lab 5-2

Lab 5-2: Plan and Create (cont'd.)

Input	Processing	Output

discount rate (10%)
shipping rate 1 (0.99)
shipping rate 2 (4.99)
amount owed
member status (Y or N)
	Processing items: none
	
	Algorithm:
	1. enter amount owed and member status
	2. if (member status is Y)
	calculate amount owed by multiplying amount owed by discount rate and then subtracting the result from amount owed
	end if
	3. if (amount owed >= 100)
	add shipping rate 1 to amount owed
	else
	add shipping rate 2 to amount owed
	end if
	4. display amount owed
		amount owed

Figure 5-25 Completed IPO chart for the Heaton Boutique program

Lab 5-2: Plan and Create (cont'd.)

discount rate	shipping rate 1	shipping rate 2	amount owed	member status
~~0.1~~	~~0.99~~	~~4.99~~	~~50.25~~	~~N~~
			~~55.24~~	
~~0.1~~	~~0.99~~	~~4.99~~	~~50.25~~	~~—~~
			~~50.22~~	
~~0.1~~	~~0.99~~	~~4.99~~	~~125.0~~	~~N~~
			~~125.99~~	
~~0.1~~	~~0.99~~	~~4.99~~	~~125.0~~	~~—~~
			~~11 .49~~	
0.1	0.99	4.99	~~125.0~~	
			125.99	

be sure to test with invalid data

no discount is given when the member status is invalid

Figure 5-26 Completed desk-check table for the Heaton Boutique algorithm

Lab 5-2: Plan and Create (cont'd.)

IPO chart information	C++ instructions
Input	
discount rate (10%)	`const double DISCOUNT_RATE = 0.1;`
shipping rate 1 (0.99)	`const double SHIP_CHG1 = 0.99;`
shipping rate 2 (4.99)	`const double SHIP_CHG2 = 4.99;`
amount owed	`double amtOwed = 0.0;`
member status (Y or N)	`char member = ' ';`
Processing	
none	
Output	
amount owed	uses the amt wed ariab dec ared abo e
Algorithm:	

1. enter amount owed and member status	`cout << "Amount owed before any discount and shipping: ";`
	`cin >> amtOwed;`
	`cout << "Premier Club member (Y/N)? ";`
	`cin >> member;`

2. i (member status is Y)
ca cu ate amount owed b mu tip
amount owed b discount rate and
then subtracting the resu t rom
amount owed
	`if (toupper(member) == 'Y')`
	`ingamtOwed -= amtOwed *`
	`DISCOUNT_RATE;`

end i	`//end if`

i (amount owed 100)
add shipping rate 1 to amount owed
e se
add shipping rate 2 to amount owed
end i
	`if (amtOwed >= 100.0)`
	`amtOwed += SHIP_CHG1;`
	`else`
	`amtOwed += SHIP_CHG2;`
	`//end if`

4. disp a amount owed	`cout << fixed << setprecision(2);` `cout << "Amount owed after any discount and shipping:" << amtOwed << endl;`

Figure 5-27 IPO chart information and C++ instructions for the Heaton Boutique problem

Lab 5-2: Plan and Create (cont'd.)

DISCOUNT_RATE	SHIP_CHG1	SHIP_CHG2	amtOwed	member
0.1	0.99	4.99	0.0	-
			0.2	N
			.2	
0.1	0.99	4.99	0.0	-
			0.2	-
			0.22	
0.1	0.99	4.99	0.0	-
			12 0	N
			12 .99	
0.1	0.99	4.99	0.0	-
			12 0	-
			11 .49	
0.1	0.99	4.99	0.0	-
			12 .0	
			12 .99	

Figure 5-28 Completed desk-check table for the Heaton Boutique program

Lab 5-2: Plan and Create (cont'd.)

```
 1 //Lab5-2.cpp - displays the total amount due
 2 //Created/revised by <your name> on <current date>
 3
 4 #include <iostream>
 5 #include <iomanip>
 6 using namespace std;
 7
 8 int main()
 9 {
10     const double DISCOUNT_RATE = 0.1;
11     const double SHIP_CHG1 = 0.99;
12     const double SHIP_CHG2 = 4.99;
13     double amtOwed = 0.0;
14     char member = ' ';
15
16     //enter input items
17     cout << "Amount owed before any discount and shipping: ";
18     cin >> amtOwed;
19     cout << "Premier Club member (Y/N)? ";
20     cin >> member;
21
22     //subtract discount, if appropriate
23     if (toupper(member) == 'Y')
24         amtOwed -= amtOwed * DISCOUNT_RATE;
25     //end if
26
27     //add shipping
28     if (amtOwed >= 100.0)
29         amtOwed += SHIP_CHG1;
30     else
31         amtOwed += SHIP_CHG2;
32     //end if
33
34     //display final amount owed
35     cout << fixed << setprecision(2);
36     cout << "Amount owed after any discount and shipping: "
37         << amtOwed << endl;
38
39     return 0;
40 }  //end of main function
```

Figure 5-29 Heaton Boutique program

Lab 5-3: Modify

- Modify the Heaton Boutique program from Lab 5-2 to give a 10% discount to members of the store's Premier Club, and a 5% discount to all other customers.

Lab 5-4: What's Missing?

- The program in this lab should display the total price of the tickets purchased by a customer. A maximum number of 10 tickets can be purchased.
- Put the C++ instructions in the proper order, and then determine the one or more missing instructions.
- Test the program three times using the following data: 8, 12, and -3.

Lab 5-5: Desk-Check

- Desk-check the code shown in Figure 5-30 using the numbers 5 and 0.
- Although the code displays the appropriate message, is it considered efficient?
- How can you make the code more efficient?

```
int quantity = 0;
cout << "Quantity: ";
cin >> quantity;

if (quantity <= 0)
    cout << "The quantity must be greater than 0." << endl;
//end if
if (quantity > 0)
    cout << "Valid quantity" << endl;
//end if
```

Figure 5-30 Code for Lab 5-5

Lab 5-6: Debug

- Follow the instructions for starting C++ and opening the Ch05-Lab5-6.cpp file.
- Test the program using codes 1, 2, and 3. Use 100 as the purchase price.
- Debug the program.