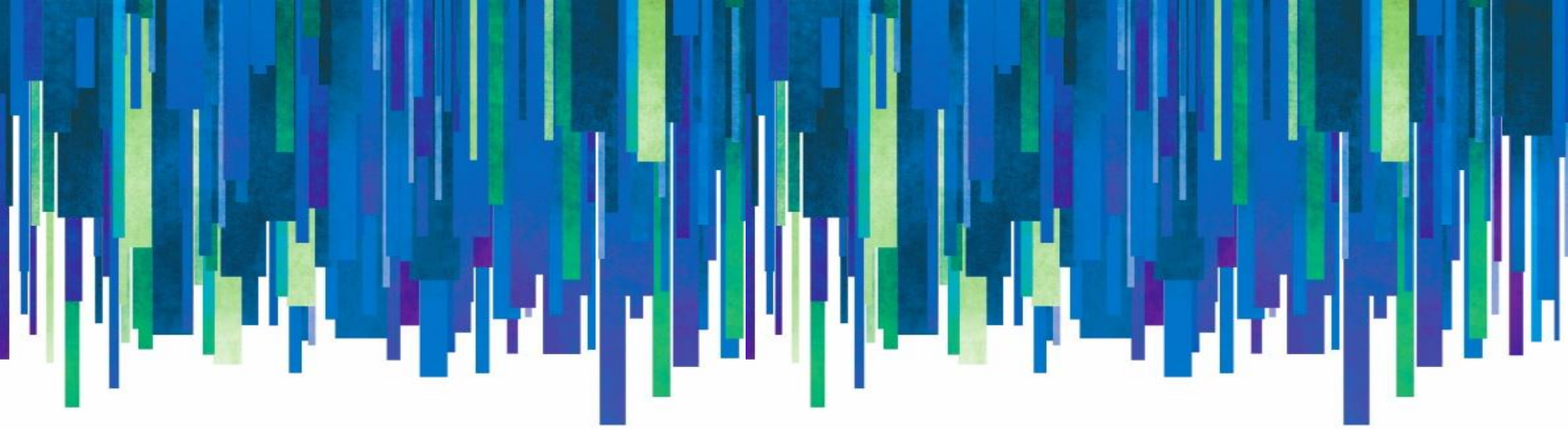


Introduction to Programming in C++
Seventh Edition

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
- 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.)

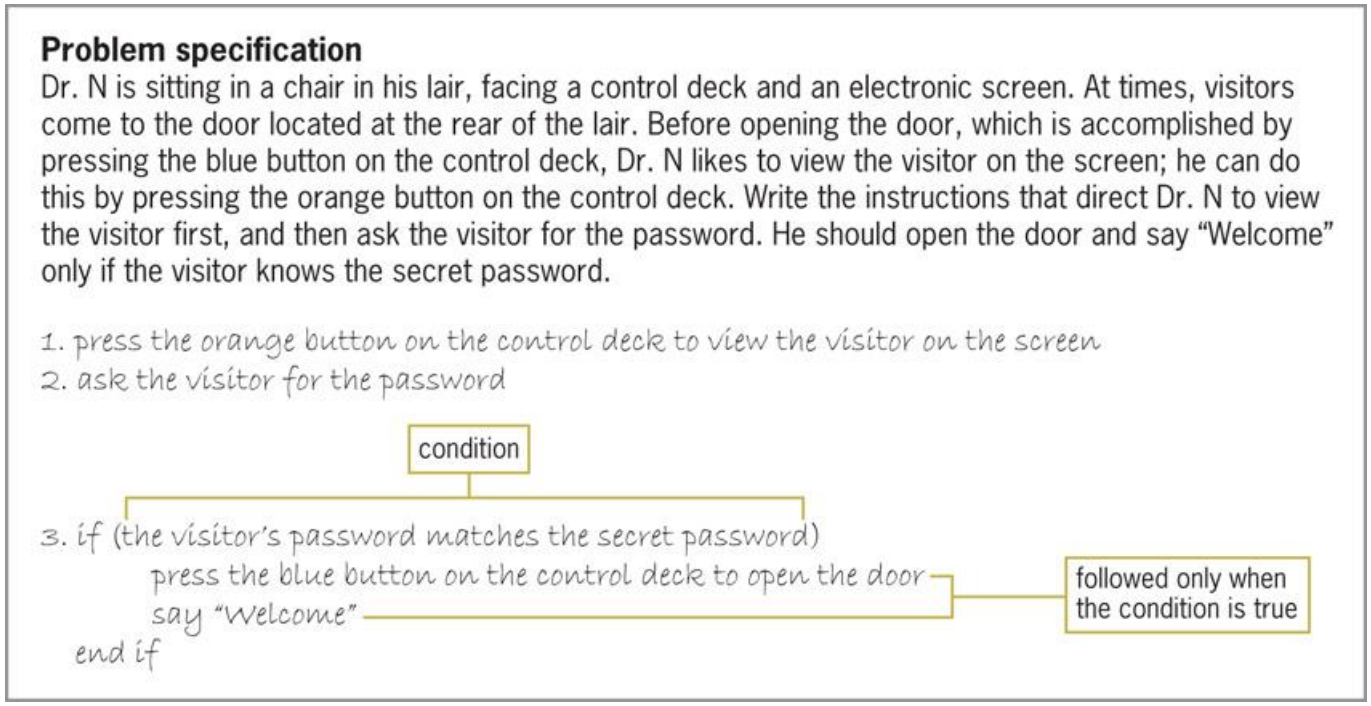


Figure 5-2 A problem that requires the sequence structure and a single-alternative selection structure

Making Decisions (cont'd.)

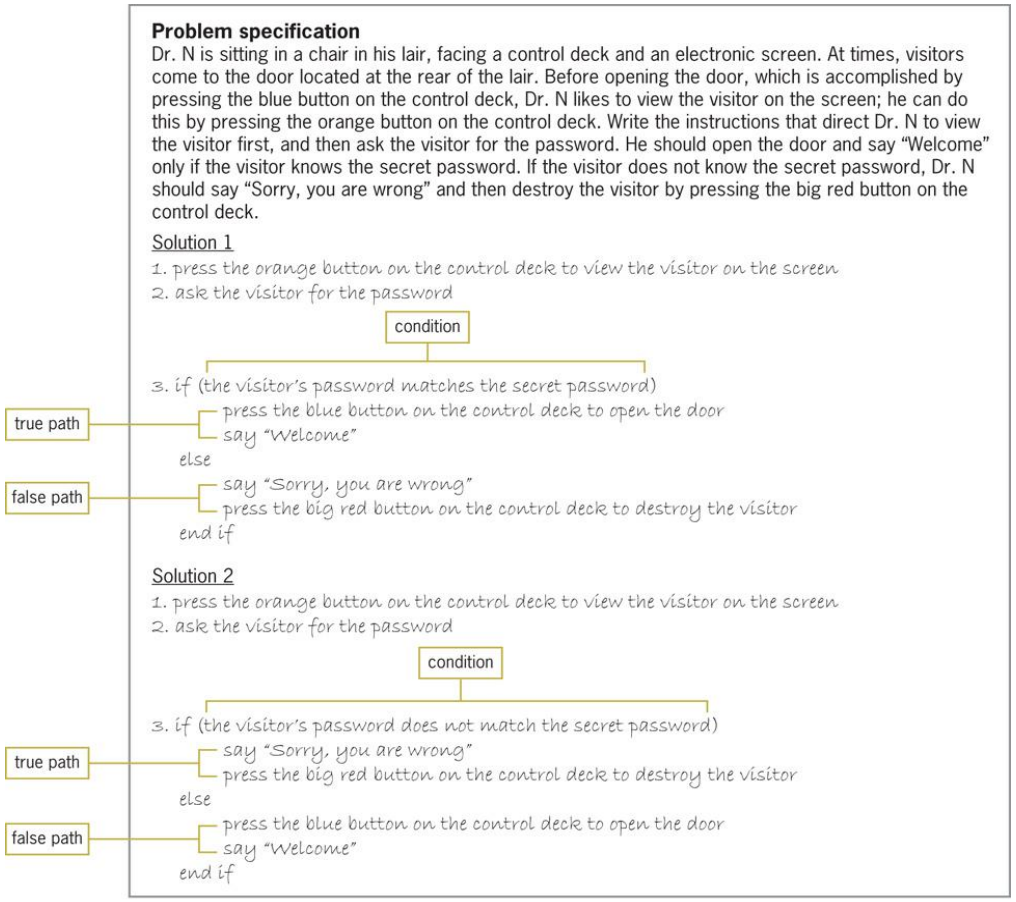


Figure 5-3 A problem that requires the sequence structure and a dual-alternative selection structure

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)

Problem specification
Jerrili's Trading Store wants a program that allows a salesclerk to enter an item's price and the quantity purchased by a customer. The store gives the customer a 10% discount when the quantity purchased is over 5. The program should calculate and display the total amount the customer owes.

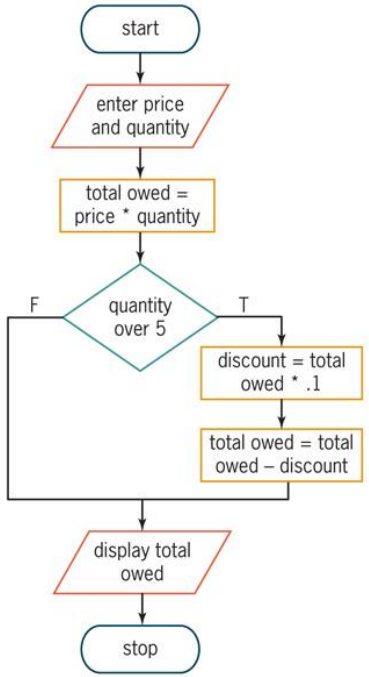


Figure 5-4 Flowchart showing a single-alternative selection structure

Flowcharting a Selection Structure (cont'd)

Problem specification
Mary Kettleson wants a program that calculates and displays her annual bonus, given her annual sales amount. Mary receives a 2% bonus when her annual sales are at least $15,000; otherwise, she receives a 1.5% bonus.

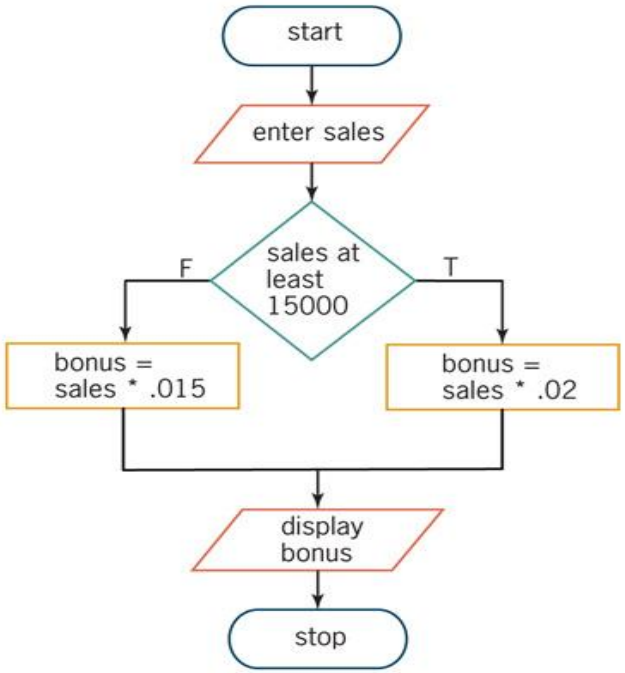


Figure 5-5 Flowchart showing a dual-alternative selection structure

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 5-6 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 5-6 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)
```

Figure 5-7 How to use comparison operators in an `if` statement's condition

Comparison Operators (cont'd.)

Original expression	`5 - 2 > 1 + 2`
The subtraction is performed first	`3 > 1 + 2`
The addition is performed next	`3 > 3`
The > comparison is performed last	`false`

Figure 5-8 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.)

IPO chart information	C++ instructions
Input	
first number	`int firstNum = 0;`
second number	`int secondNum = 0;`
Processing	
none	
Output	
first number (lowest)	
second number (highest)	
Algorithm	

1. enter the first number and the second number	`cout << "Enter an integer: ";`
	`cin >> firstNum;`
	`cout << "Enter another integer: ";`
	`cin >> secondNum;`

2. if (the first number is greater than the second number)	`if (firstNum > secondNum)`

swap the numbers so that the first number is the lowest number	`{`
	`int temp = 0;`
	`temp = firstNum;`
	`firstNum = secondNum;`
	`secondNum = temp;`

end if	`} //end if`
3. display the first number and the second number	`cout << "Lowest: " << firstNum << endl;` `cout << "Highest: " << secondNum << endl;`

Figure 5-9 IPO chart information and C++ instructions for the swapping program

Swapping Numeric Values (cont'd.)

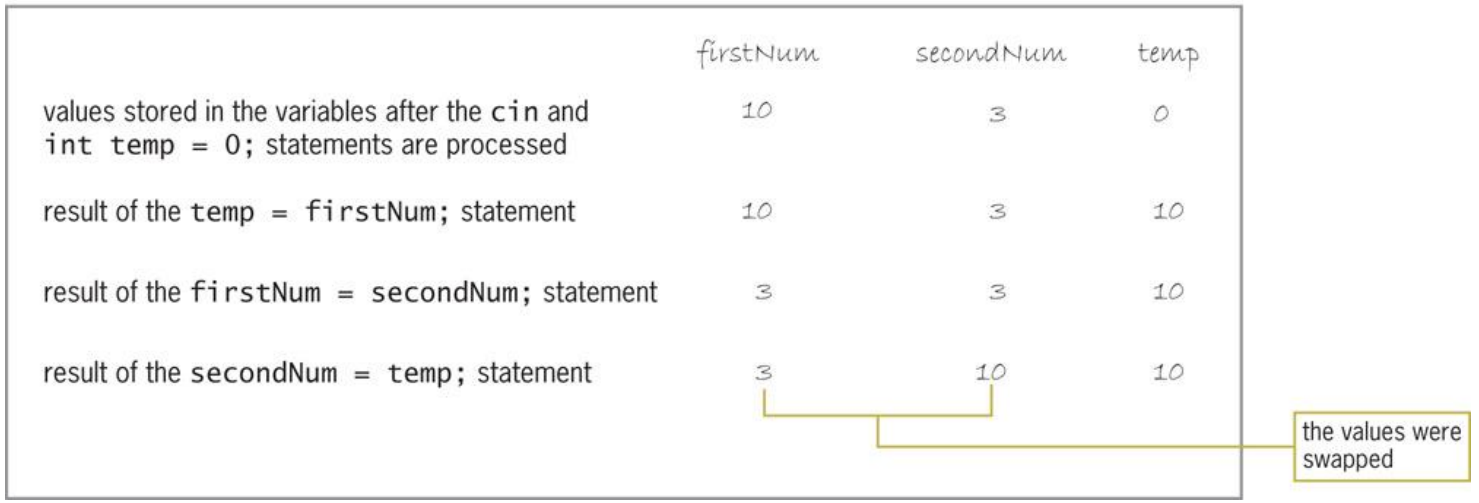


Figure 5-10 Illustration of the swapping concept

Swapping Numeric Values (cont'd.)

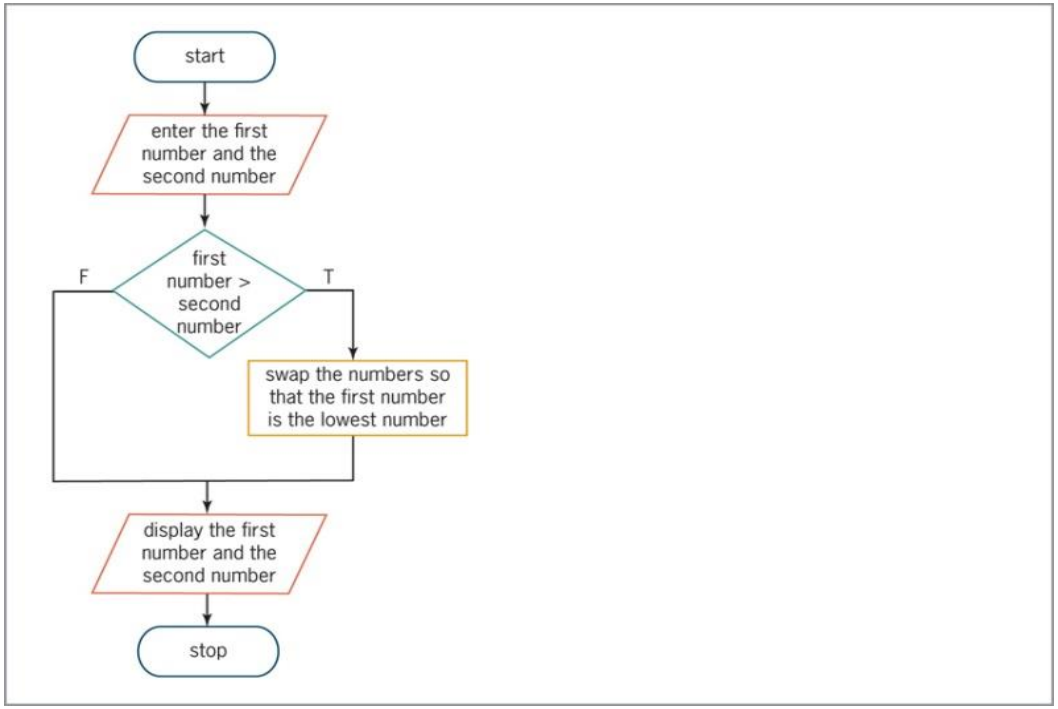


Figure 5-11 Flowchart for the swapping program

Swapping Numeric Values (cont'd.)

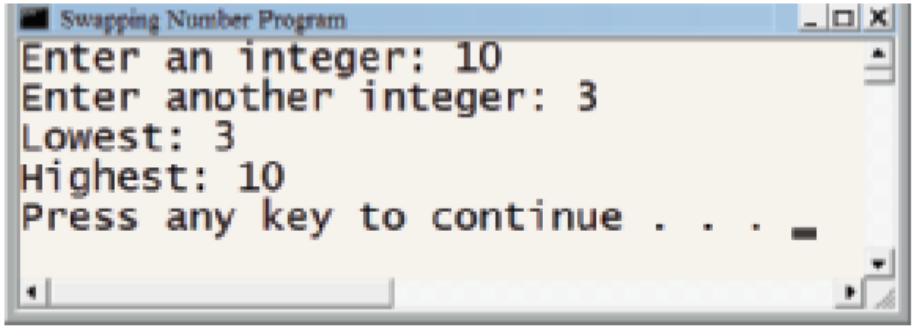


Figure 5-12 Sample run of the swapping program

Displaying the Sum or Difference

- Example program (next slide) displays the sum or difference of two numbers entered by the user
- Choice of addition or subtraction is entered by user as well
- Uses a dual-alternative selection structure

Displaying the Sum or Difference (cont'd.)

IPO chart information	C++ instructions
Input	
operation	`int operation = 0;`
January sales	`double janSales = 0.0;`
February sales	`double febSales = 0.0;`
Processing	
none	
Output	
either the sum of or the difference between the January sales and February sales	`double answer = 0.0;`
Algorithm	

1. enter the operation, January sales, and February sales	`cout << "Enter 1 (add) or 2 (subtract): ";`
	`cin >> operation;`
	`cout << "January sales: ";`
	`cin >> janSales;`
	`cout << "February sales: ";`
	`cin >> febSales;`

2. if (the operation is 1)	`if (operation == 1)`
calculate the answer by adding the January sales to the February sales	`{` `answer = janSales + febSales;`
display "Sum:" and the answer	`cout << "Sum: " << answer << endl;`
else	`}` `else`
calculate the answer by subtracting the February sales from the January sales	`{` `answer = janSales - febSales;`
display "Difference:" and the answer	`cout << "Difference: " << answer << endl;`
end if	`} //end if`

Figure 5-13 IPO chart information and C++ instructions for the sum or difference program

Displaying the Sum or Difference (cont'd.)

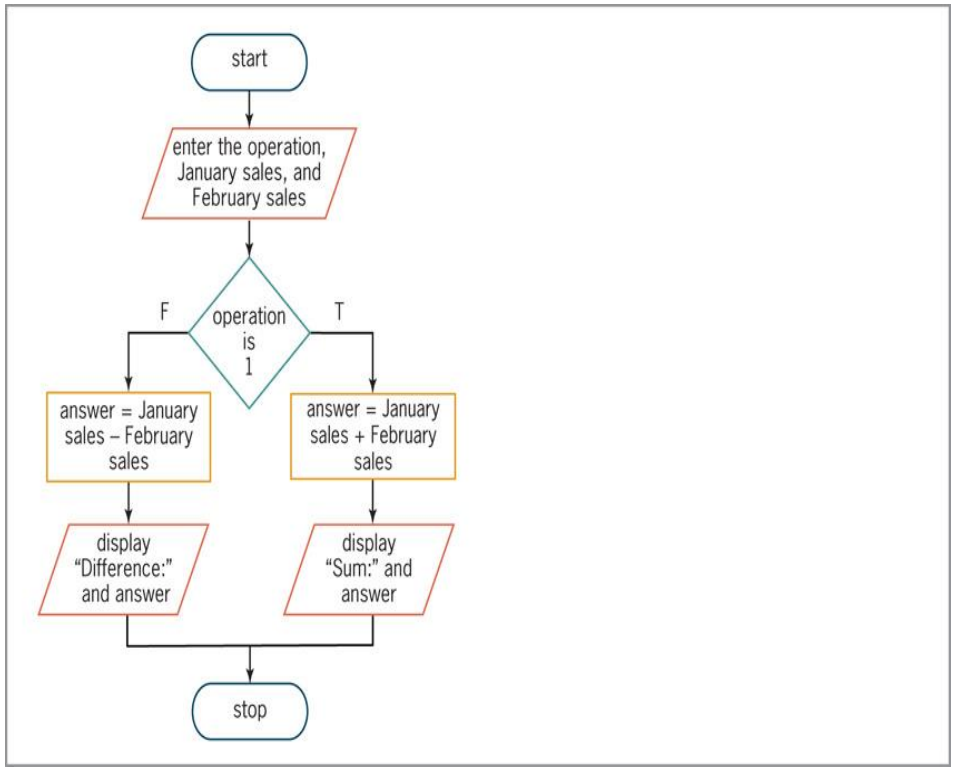


Figure 5-14 Flowchart for sum or difference program

Displaying the Sum or Difference (cont'd.)

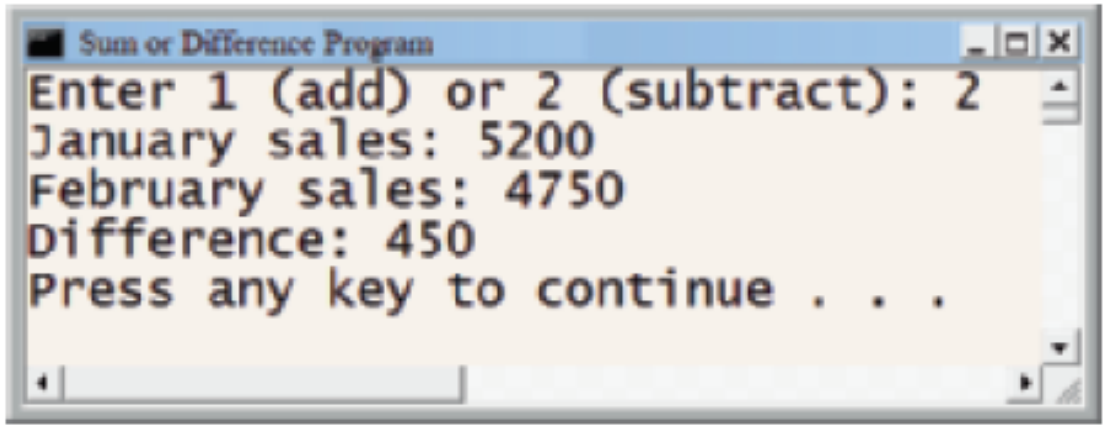


Figure 5-15 Sample run of the sum or difference program

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* sub-conditions must be true for the compound condition to evaluate to true	1
Or (||)	only *one* of the sub-conditions needs to be true for the compound condition to evaluate to true	2

(continues)

Figure 5-16 How to use logical operators in an `if` statement's condition

Logical Operators (cont'd.)

(continued)

Example 1

```
int quantity = 0;
cin >> quantity;
if (quantity > 0 && quantity < 50)
```

The compound condition evaluates to true when the number stored in the `quantity` variable is greater than 0 and, at the same time, less than 50; 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.

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 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 5-16 How to use logical operators in an `if` statement's condition (cont'd)

Logical Operators (cont'd.)

Truth table for the And (&&) operator

sub-condition1	sub-condition2	sub-condition1 && sub-condition2
true	true	true
true	false	false
false	true (not evaluated)	false
false	false (not evaluated)	false

Truth table for the Or (||) operator

sub-condition1	sub-condition2	sub-condition1 || sub-condition2
true	true (not evaluated)	true
true	false (not evaluated)	true
false	true	true
false	false	false

Figure 5-17 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

```
//declare constant and variables
const int PAY_RATE = 10;
int hoursWorked    = 0;
int grossPay       = 0;

//enter input items
cout << "Hours worked (0 through 40): ";
cin >> hoursWorked;

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

Example 2

```
//declare constant and variables
const int PAY_RATE =  10;
int hoursWorked    =  0;
int grossPay       =  0;

//enter input items
cout << "Hours worked (0 through 40): ";
cin >> hoursWorked;

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

Figure 5-18 Examples of C++ instructions for the gross pay program

Calculating Gross Pay (cont'd.)

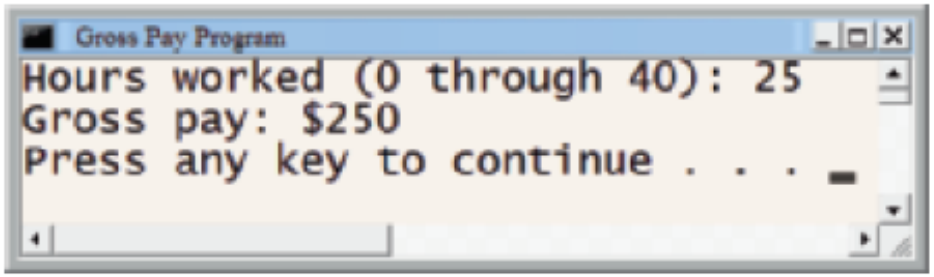


Figure 5-19 First sample run of the gross pay program's code

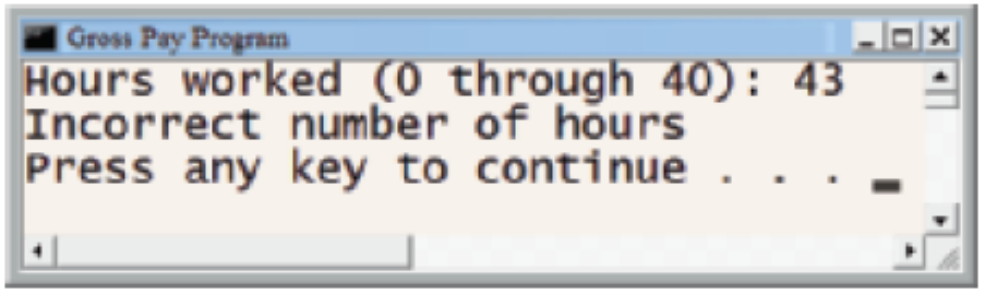
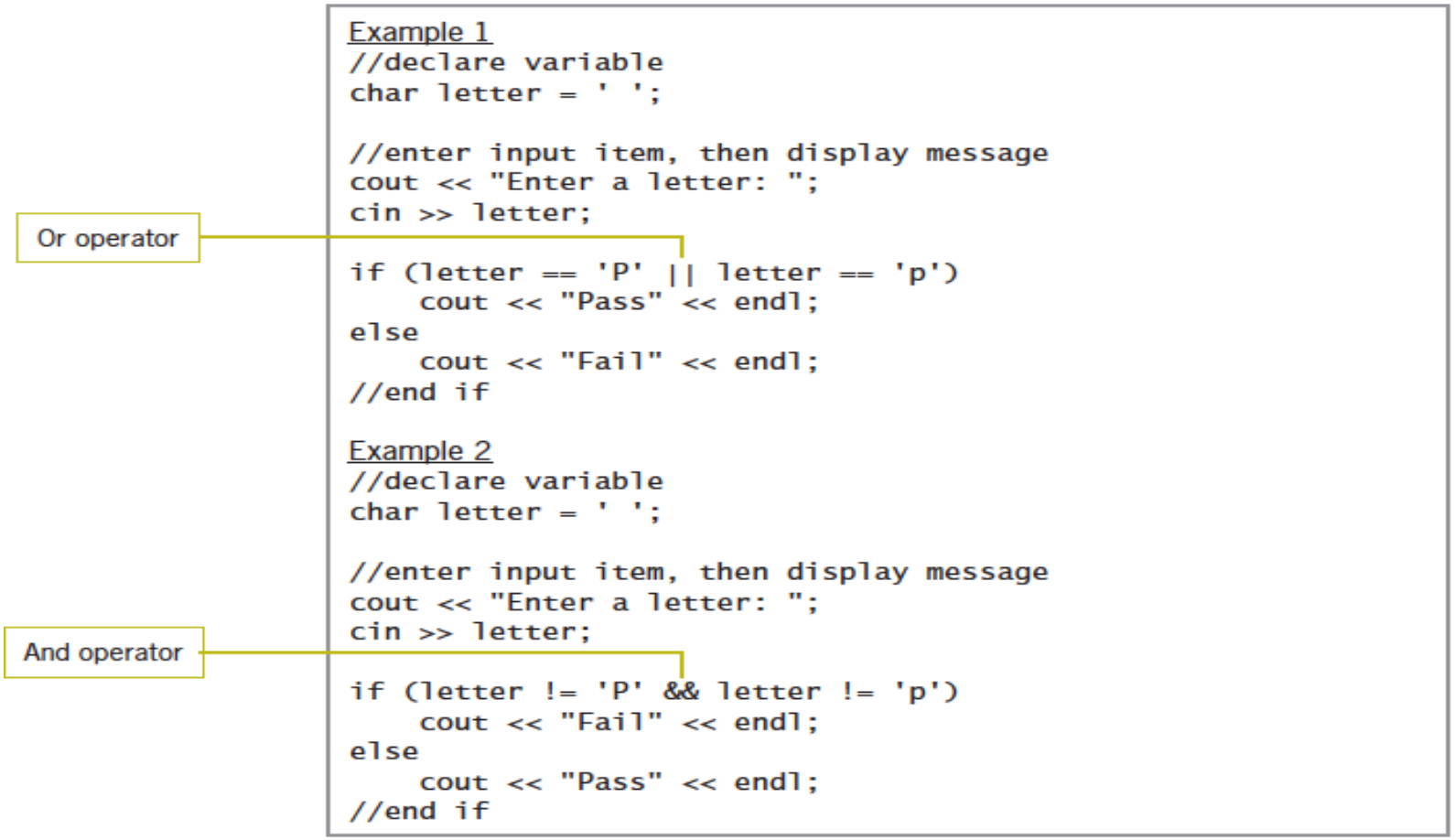


Figure 5-20 Second sample run of the gross pay program's code

Pass/Fail Program

- Example problem description is given in which program must output “Pass” if user enters ‘P’ or ‘p’, and “Fail” otherwise
- Character comparisons are case sensitive in C++
- Program must check separately whether the user entered ‘P’ or ‘p’
- Dual-alternative selection structure is used to implement program
- Compound condition with Or operator is used to perform check

Pass/Fail Program (cont'd)



Example 1

```
//declare variable
char letter = ' ';

//enter input item, then display message
cout << "Enter a letter: ";
cin >> letter;

if (letter == 'P' || letter == 'p')
    cout << "Pass" << endl;
else
    cout << "Fail" << endl;
//end if
```

Example 2

```
//declare variable
char letter = ' ';

//enter input item, then display message
cout << "Enter a letter: ";
cin >> letter;

if (letter != 'P' && letter != 'p')
    cout << "Fail" << endl;
else
    cout << "Pass" << endl;
//end if
```

Figure 5-21 Examples of C++ instructions for the Pass/Fail program

Pass/Fail Program (cont'd.)

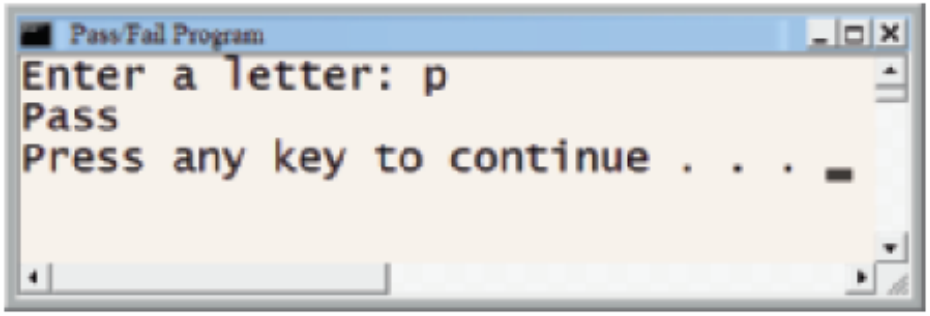


Figure 5-22 First sample run of the Pass/Fail program's code

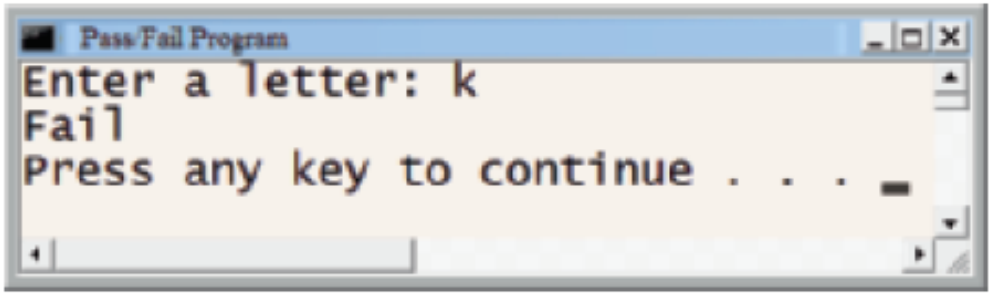


Figure 5-23 Second sample run of the Pass/Fail program's code

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 sub-conditions must be true for the compound condition to evaluate to true	7
Or (||)	only one of the sub-conditions needs to be true for the compound condition to evaluate to true	8

Example

Original expression	30 > 75 / 3 && 5 < 10 * 2
75 / 3 is performed first	30 > 25 && 5 < 10 * 2
10 * 2 is evaluated second	30 > 25 && 5 < 20
30 > 25 is evaluated third	true && 5 < 20
5 < 20 is evaluated fourth	true && true
true && true is evaluated last	true

Figure 5-24 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(letter) == 'P')
```

The condition compares the uppercase character returned by the `toupper` function with the uppercase letter P. The condition evaluates to true when the character stored in the `letter` variable is either P or p.

Example 2

```
if (tolower(letter) == 'p')
```

The condition compares the lowercase character returned by the `tolower` function with the lowercase letter p. The condition evaluates to true when the character stored in the `letter` variable is either P or p.

Example 3

```
initial = toupper(initial);
```

The assignment statement changes the contents of the `initial` variable to uppercase.

Figure 5-25 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 — Result

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

displays 10575.250000

Example 2

```
double rate = 5.12345623;
cout << fixed << rate << endl;
```

displays 5.123456

Example 3

```
double rate = 5.123456932;
cout << fixed << rate << endl;
```

displays 5.123457

Example 4

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

displays 1.057525e+004

Figure 5-26 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	Result

`double sales = 3500.6;`
`cout << fixed;`
`cout << setprecision(2);`
`cout << sales << endl;`
	displays 3500.60

Example 2	

`double rate = 10.0732;`
`cout << fixed << setprecision(3);`
`cout << rate << endl;`
	displays 10.073

Example 3	

`double sales = 3467.55;`
`cout << fixed;`
`cout << setprecision(0) << sales;`
	displays 3468

Figure 5-27 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 an employee's new salary
 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 salary = 0.0;
11     double rate   = 0.0;
12     char payGrade = ' ';
13
14     cout << "Current salary: ";
15     cin >> salary;
16     cout << "Pay grade (1, 2, or 3): ";
17     cin >> payGrade;
18
19     if (payGrade == '1')
20         rate = .03;
21     else
22         rate = .02;
23     //end if
24
25     salary = salary + salary * rate;
26     cout << fixed << setprecision(2);
27     cout << "New salary: " << salary << endl;
28
29     system("pause");
30     return 0;
31  }  //end of main function
```

Figure 5-28 Program for Lab 5-1

Lab 5-2: Plan and Create

- Plan and create an algorithm for the manager of the Willow Springs Health Club

The manager of Willow Springs Health Club wants a program that allows her to enter the number of calories and grams of fat contained in a specific food. The program should calculate and display two values: the food's fat calories and its fat percentage. The fat calories are the number of calories attributed to fat. The fat percentage is the ratio of the food's fat calories to its total calories. You can calculate a food's fat calories by multiplying its fat grams by the number 9, because each gram of fat contains 9 calories. To calculate the fat percentage, you divide the food's fat calories by its total calories and then multiply the result by 100. The fat percentage should be displayed with zero decimal places. The program should display an appropriate error message if either or both input values are less than zero.

Figure 5-29 Problem specification for Lab 5-2

Lab 5-3: Modify

- Currently, the `if` statement's true path in Lab 5-2 handles valid data, while its false path handles invalid data
- Modify the program so that invalid data is handled in the true path, and valid data is handled in the false path

Lab 5-4: Desk-Check

- Desk-check the code shown in Figure 5-34 using the letter ‘P’
- Why is it inefficient? How could you improve it?

```
//declare variable
char letter = ' ';

//enter input item, then display message
cout << "Enter a letter: ";
cin >> letter;

if (letter == 'P' || letter == 'p')
    cout << "Pass" << endl;
//end if
if (letter != 'P' || letter != 'p')
    cout << "Fail" << endl;
//end if
```

Figure 5-35 Code for Lab 5-4

Lab 5-5: Debug

- Follow the instructions for starting C++ and opening the Lab5-5.cpp file
- Test the program using codes 1, 2, and 3
- Debug the program