An Introduction to Programming with C++

Fifth Edition

Chapter 5

The Selection Structure

Objectives

- Write pseudocode for the selection structure
- Create a flowchart for the selection structure
- Code the `if` and `if/else` forms of the selection structure
- Write code that uses comparison operators and logical operators

Objectives (continued)

- Convert the contents of a `char` variable to uppercase or lowercase
- Convert the contents of a `string` variable to uppercase or lowercase
- Use the .NET `ToUpper()`, `ToLower()`, and `CompareTo()` methods
- Format numeric output in .NET

Concept Lesson

- Using the Selection Structure
- Writing Pseudocode for the `if` and `if/else` Selection Structures
- Flowcharting the `if` and `if/else` Selection Structures
- Coding the `if` and `if/else` Selection Structures

Concept Lesson (continued)

- Comparison Operators
- Logical Operators
- Comparing Characters
- Comparing Strings

Using the Selection Structure

- Also called the **decision structure**
  - **Condition** specifies decision
    - Results in either a true or false answer only
  - Three forms: `if`, `if/else`, and `switch` (or `case`)

Using the Selection Structure (continued)

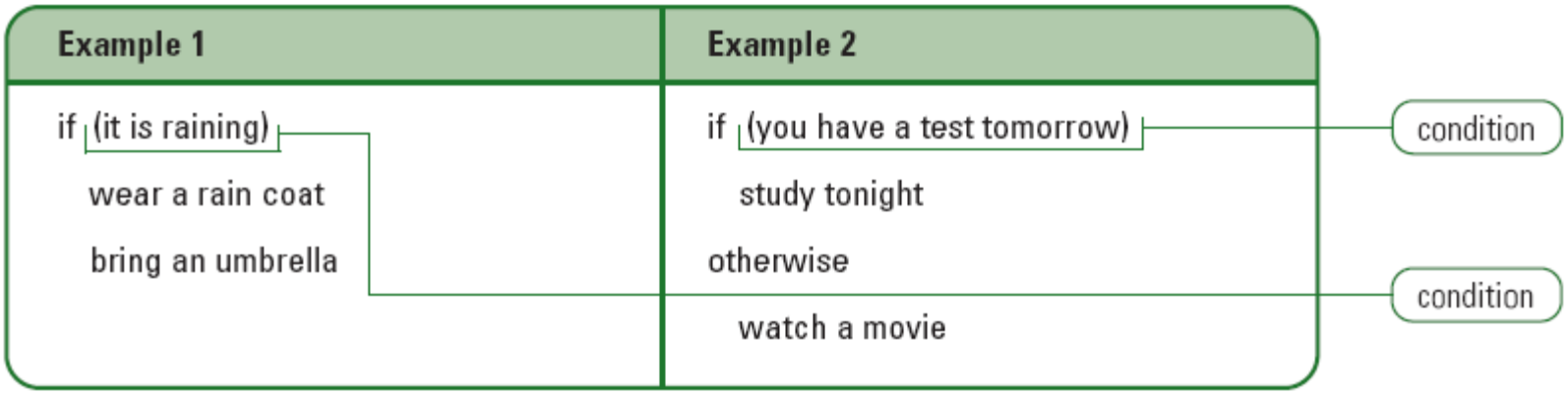


Figure 5-1: Selection structures you might use today

Writing Pseudocode for the `if` and `if/else` Selection Structures

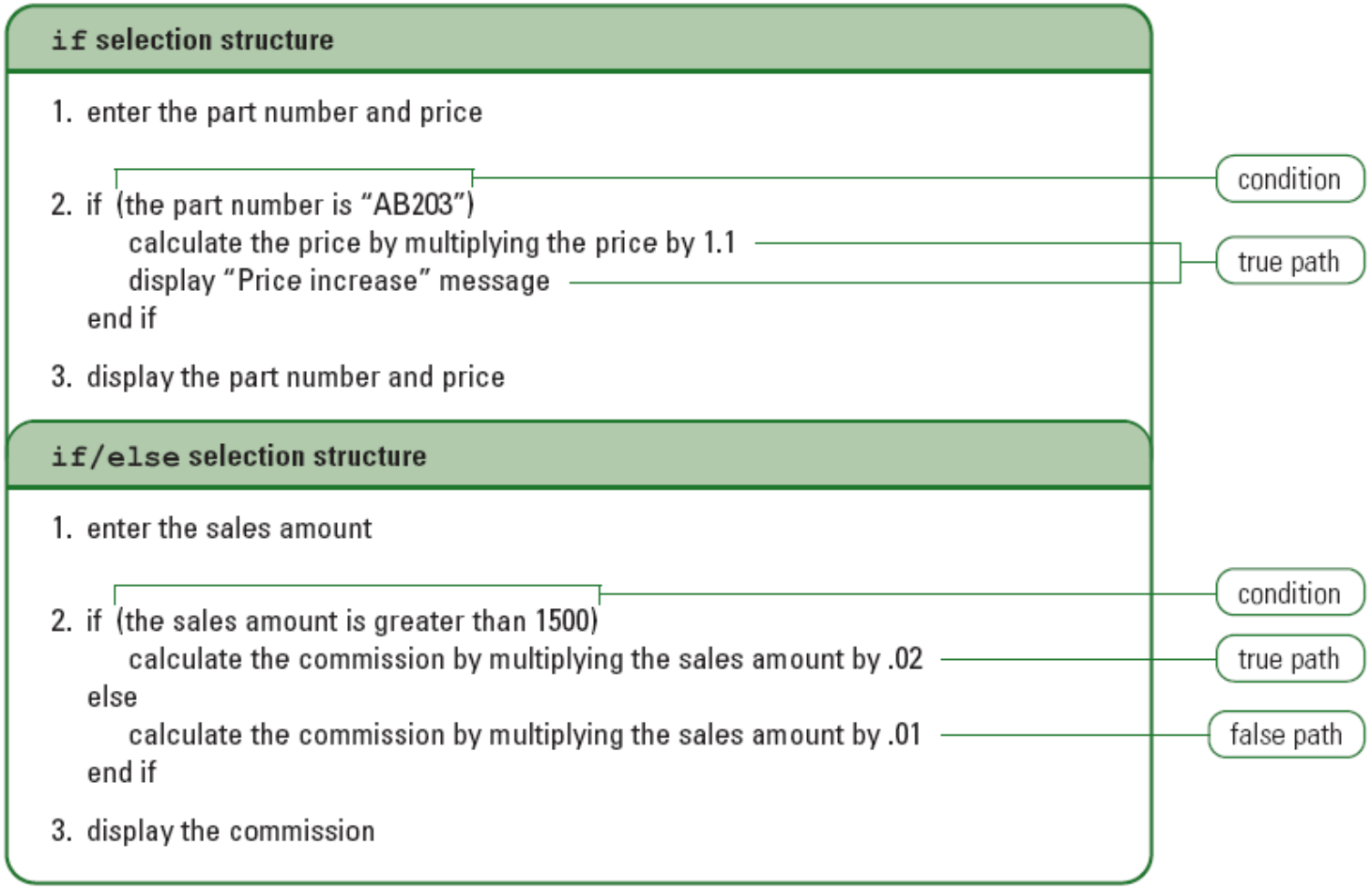
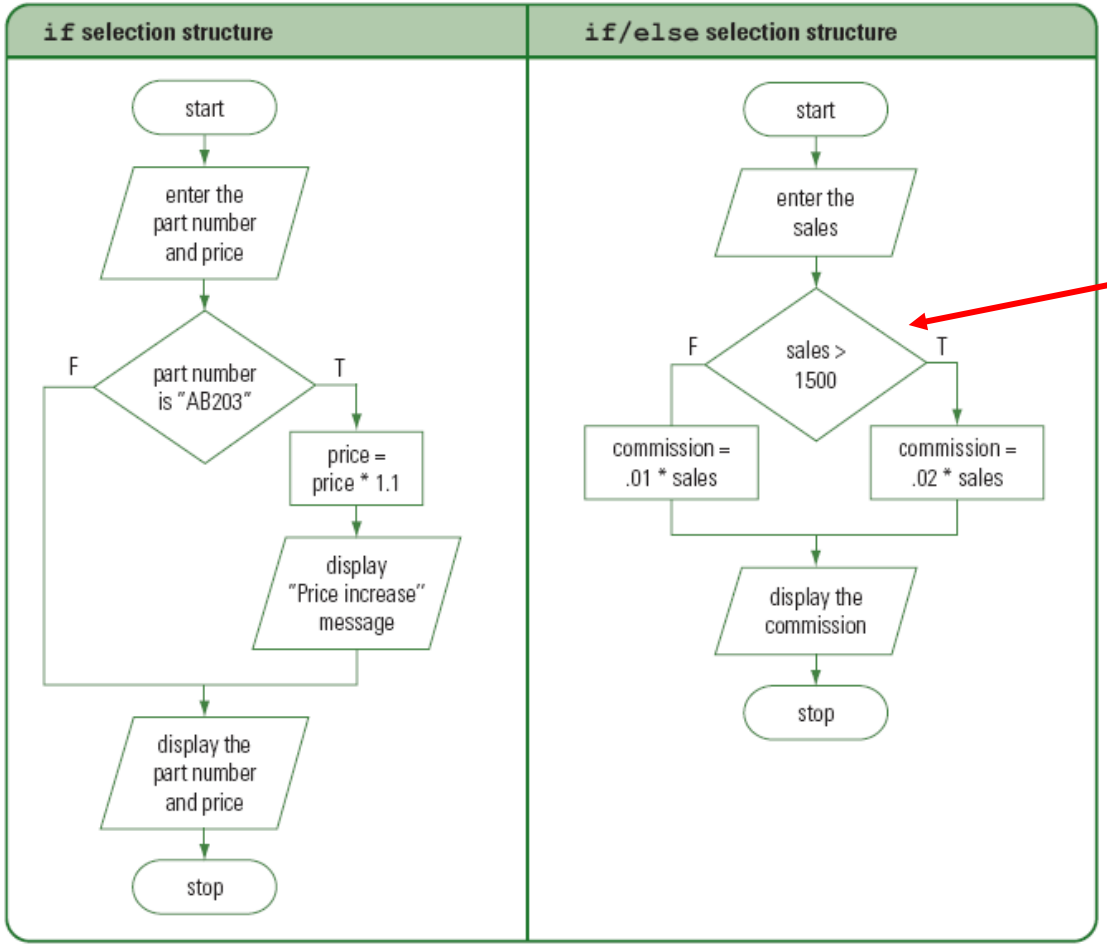


Figure 5-2: Examples of the `if` and `if/else` selection structures written in pseudocode

Flowcharting the `if` and `if/else` Selection Structures



selection/repetition symbol

Figure 5-3: Examples of the `if` and `if/else` selection structures drawn in flowchart form

Coding the `if` and `if/else` Selection Structures

Use the `if` Statement to Code the `if` and `if/else` Selection Structures

Syntax

```
if (condition)
      one statement, or a block of statements enclosed in braces, to be processed
      when the condition is true
[else
      one statement, or a block of statements enclosed in braces, to be processed
      when the condition is false]
//end if
```

Example 1 – `if` form with one statement

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

Figure 5-4 How to use the `if` statement to code the `if` and `if/else` selection structures

Coding the `if` and `if/else` Selection Structures (continued)

Example 2 – `if` form with multiple statements

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

Example 3 – `if/else` form with one statement in each path

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

Example 4 – `if/else` form with multiple statements in true path and one statement in false path

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

Example 5 – `if/else` form with one statement in true path and multiple statements in false path

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

Example 6 – `if/else` form with multiple statements in both paths

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

Figure 5-4 How to use the `if` statement to code the `if` and `if/else` selection structures *(Continued)*

Comparison Operators

- Often called **relational operators**
- If expression has multiple comparison operators at same precedence, it is evaluated from left to right
- Comparison operators are evaluated after any arithmetic operators in the expression

Comparison Operators (continued)

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

In the examples, `quantity`, `age`, `onhand`, and `target` are `int` variables.

Examples:

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

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

Comparison Operators (continued)

Evaluation steps	Result
Original expression	5 – 2 > 1 + 2
5 – 2 is evaluated first	3 > 1 + 2
1 + 2 is evaluated second	3 > 3
3 > 3 is evaluated last	false

Figure 5-6: Evaluation steps for an expression containing arithmetic and comparison operators

Comparison Operator Program 1: Swapping Numerical Values

Pseudocode – Swapping values program

1. enter the first number and the second number
2. if (the first number is greater than the second number)
   swap the numbers so that the first number is the lowest number
   end if
3. display the first number (the lowest) and the second number (the highest)

Figure 5-7: Pseudocode for the swapping values program

Comparison Operator Program 1: Swapping Numerical Values (continued)

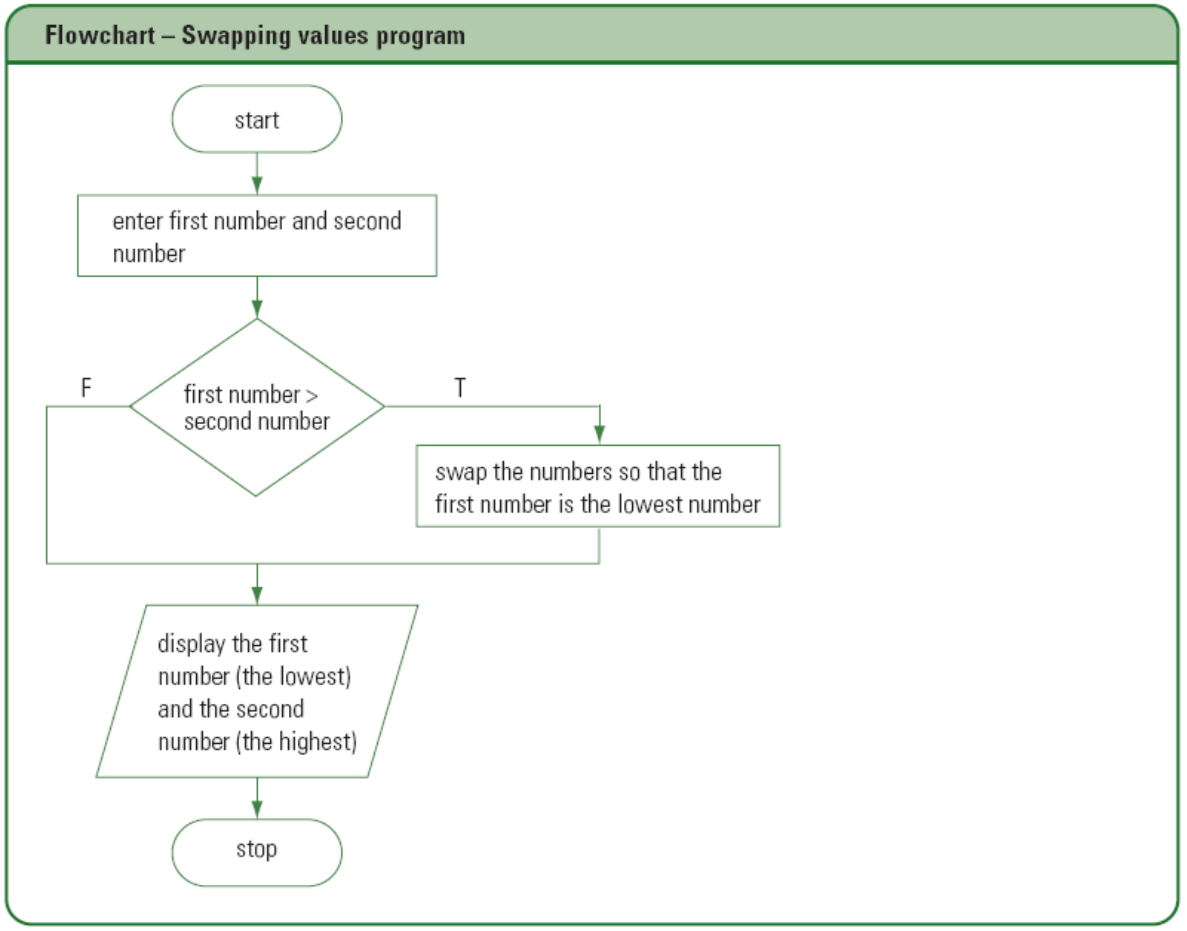


Figure 5-8: Flowchart for the swapping values program

Comparison Operator Program 1: Swapping Numerical Values (continued)

C++ code – Swapping values program

```
int main()
{
     //declare variables
     int first  = 0;
     int second = 0;

     //enter input items
     cout << "Enter the first number: ";
     cin >> first;
     cout << "Enter the second number: ";
     cin >> second;

     //if necessary, swap values
     if (first > second)
     {
          int temp = 0;
          temp = first;
          first = second;
          second = temp;
     }   //end if

     //display output items
     cout << endl;
     cout << "Lowest number: " << first << endl;
     cout << "Highest number: " << second << endl;

     return 0;
}    //end of main function
```

true path statement block

condition

temp is a **local variable**

Figure 5-9: C++ code for the swapping values program

Comparison Operator Program 1: Swapping Numerical Values (continued)

Figure 5-10: Sample run of the swapping values program

Comparison Operator Program 1: Swapping Numerical Values (continued)

	temp	first	second
values stored in the variables after the `cin` statements are processed	0	8	4
result of the `temp = first;` statement	8	8	4
result of the `first = second;` statement	8	4	4
result of the `second = temp;` statement, which completes the swapping process	8	4	8

values were swapped

Figure 5-11: Illustration of the swapping process

Comparison Operator Program 2: Displaying the Sum or Difference

Pseudocode – Displaying sum or difference program

```
1. enter the operation, first number, and second number
2. if (the operation is equal to 1)
       calculate the sum by adding together the first number and the second number
       display the sum
   else
       calculate the difference by subtracting the second number from the first number
       display the difference
   end if
```

Figure 5-12: Pseudocode for the displaying sum or difference program

Comparison Operator Program 2: Displaying the Sum or Difference (continued)

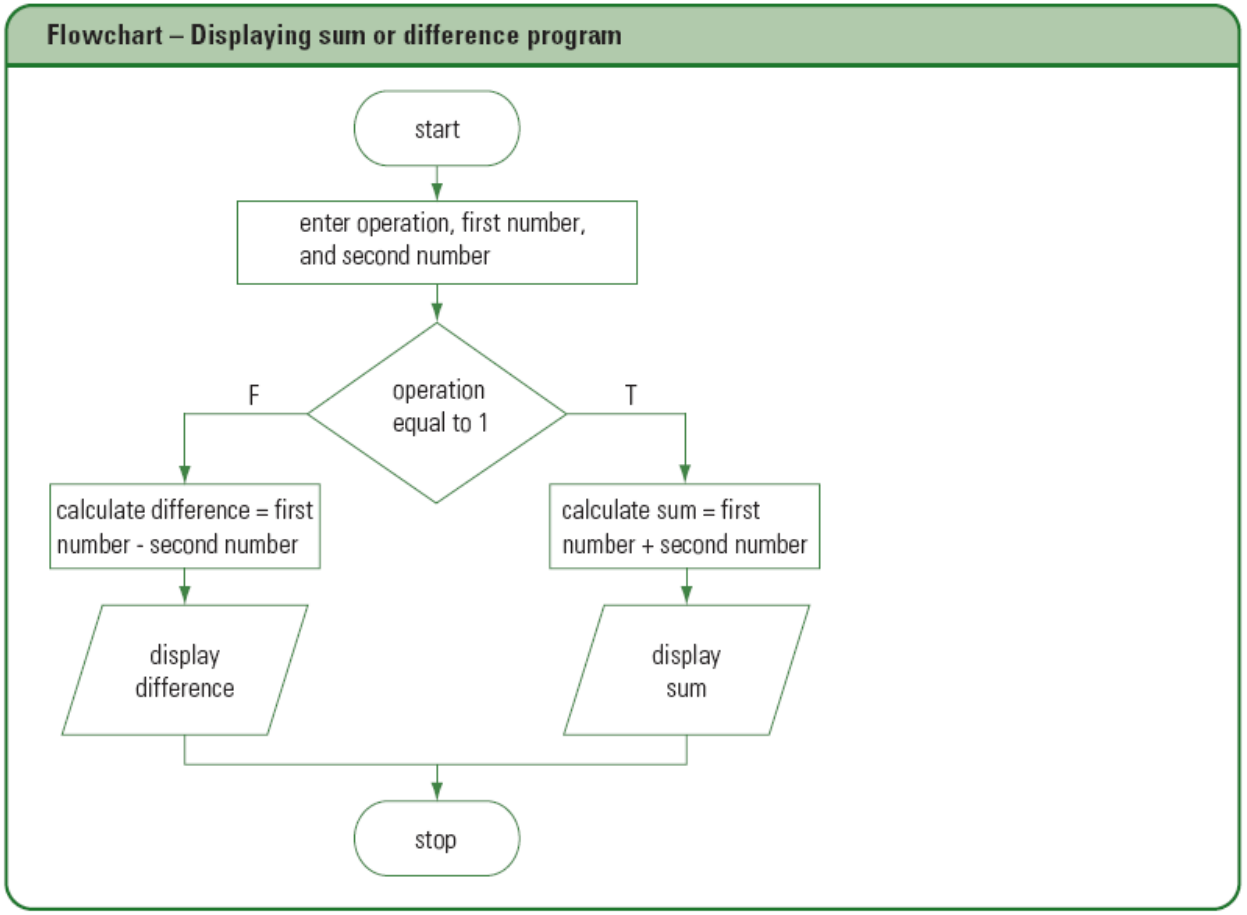


Figure 5-13: Flowchart for the displaying sum or difference program

Comparison Operator Program 2: Displaying the Sum or Difference (continued)

C++ code – Displaying sum or difference program

```
int main()
{
     //declare variables
     int operation = 0;
     int num1      = 0;
     int num2      = 0;
     int answer    = 0;

     //enter input items
     cout << "Enter 1 (add) or 2 (subtract): ";
     cin >> operation;
     cout << endl;
     cout << "Enter the first number:  ";
     cin >> num1;
     cout << "Enter the second number: ";
     cin >> num2;
     cout << endl;

     //calculate and display output
     if (operation == 1)
     {
          answer = num1 + num2;
          cout << "Sum: " << answer << endl;
     }
     else
     {
          answer = num1 - num2;
          cout << "Difference: " << answer << endl;
     }    //end if

     return 0;
}    //end of main function
```

condition

Figure 5-14: C++ code for the displaying sum or difference program

Comparison Operator Program 2: Displaying the Sum or Difference (continued)

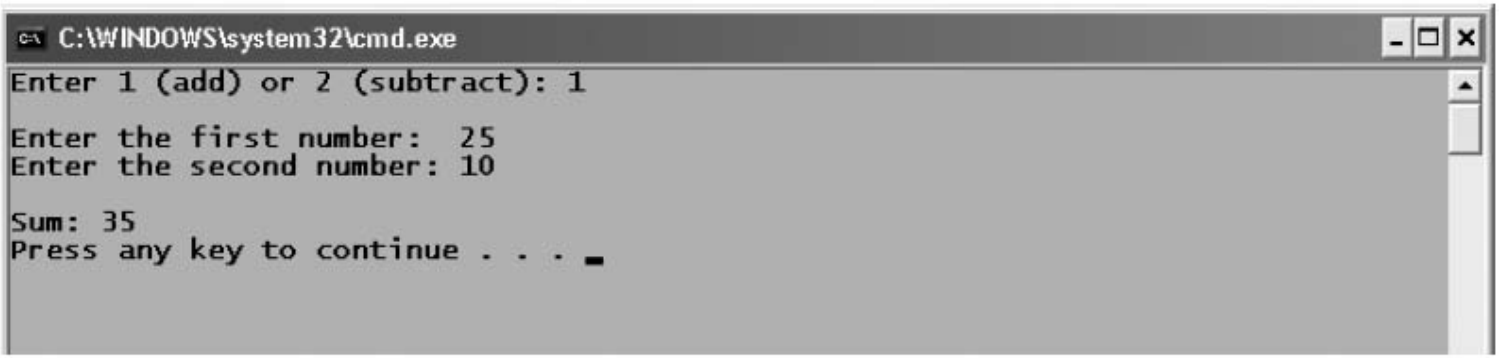


Figure 5-15: Sample run of the displaying sum or difference program

Logical Operators

- **Logical operators** allow you to combine two or more *conditions* into one compound *condition*
  - Sometimes called **Boolean operators**
- And/Or operators are evaluated after any arithmetic or comparison operators in an expression

Logical Operators (continued)

Use Logical Operators in an `if` Statement's *Condition*

Operator	Operation	Precedence number
And (`&&`)	all conditions connected by the And operator must be true for the compound condition to be true	1
Or (`||`)	only one of the conditions connected by the Or operator needs to be true for the compound condition to be true	2

In the examples, `quantity` and `age` are `int` variables, and `price` is a `double` variable.

Examples:

```
if (quantity > 0 && quantity < 50)
if (age < 21 || age > 55)
if (quantity < 100 && price < 10.35)
```

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

Logical Operators (continued)

Truth table for the && (And) operator

value of *condition1*	value of *condition2*	value of *condition1* && *condition2*
true	true	true
true	false	false
false	true	false
false	false	false

Truth table for the || (Or) operator

value of *condition1*	value of *condition2*	value of *condition1* || *condition2*
true	true	true
true	false	true
false	true	true
false	false	false

Figure 5-17: Truth tables for the And and Or logical operators

use **short-circuit evaluation**

Using the Truth Tables

- To receive a bonus, a salesperson must be rated A and he/she must sell more than $10,000 in product

```
rating == 'A' && sales > 10000
```

- To send a letter to all A-rated salespeople and all B-rated salespeople

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

Logical Operators (continued)

Operator	Operation	Precedence number
()	overrides all other normal precedence rules	1
–	negation	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 conditions connected by the And operator must be true for the compound condition to be true	7
|| (Or)	only one of the conditions connected by the Or operator needs to be true for the compound condition to be true	8

Figure 5-18: Order of precedence for arithmetic, comparison, and logical operators in a C++ expression

Logical Operators (continued)

Evaluation steps	Result
Original expression	12 > 0 && 12 < 10 * 2
10 * 2 is evaluated first	12 > 0 && 12 < 20
12 > 0 is evaluated second	true && 12 < 20
12 < 20 is evaluated third	true && true
true && true is evaluated last	true

Figure 5-19: Evaluation steps for an expression containing arithmetic, comparison, and logical operators

Logical Operator Program: Calculating Gross Pay

Example 1: using the && (And) operator

```
int main()
{
     //declare constant
     const double PAYRATE = 10.65;

     //declare variables
     double hours = 0.0;
     double gross = 0.0;
```

Figure 5-20: C++ code showing the And and Or logical operators in the `if` statement's *condition*

Logical Operator Program: Calculating Gross Pay (continued)

data validation →

```
    //enter input items
    cout << "Enter hours worked: ";
    cin >> hours;

    //calculate and display output
    if (hours >= 0.0 && hours <= 40.0)
    {
        gross = hours * PAYRATE;
        cout << fixed << setprecision(2);
        cout << "Gross pay: " << gross << endl;
    }
    else
        cout << "Input error" << endl;
    //end if

    return 0;
}   //end of main function
```

Example 2: using the || (Or) operator

```
int main()
{
    //declare constant
    const double PAYRATE = 10.65;

    //declare variables
    double hours = 0.0;
    double gross = 0.0;

    //enter input items
    cout << "Enter hours worked: ";
    cin >> hours;

    //calculate and display output
    if (hours < 0.0 || hours > 40.0)
        cout << "Input error" << endl;
    else
    {
        gross = hours * PAYRATE;
        cout << fixed << setprecision(2);
        cout << "Gross pay: " << gross << endl;
    }   //end if

    return 0;
}   //end of main function
```

Figure 5-20: C++ code showing the And and Or logical operators in the `if` statement's *condition (Continued)*

Logical Operator Program: Calculating Gross Pay (continued)

Figure 5-21: First sample run of either of the programs shown in Figure 5-20

Figure 5-22: Second sample run of either of the programs shown in Figure 5-20

Comparing Characters

- Display the word “Pass” if user enters the letter P (uppercase or lowercase)
- Display “Fail” if user enters anything else

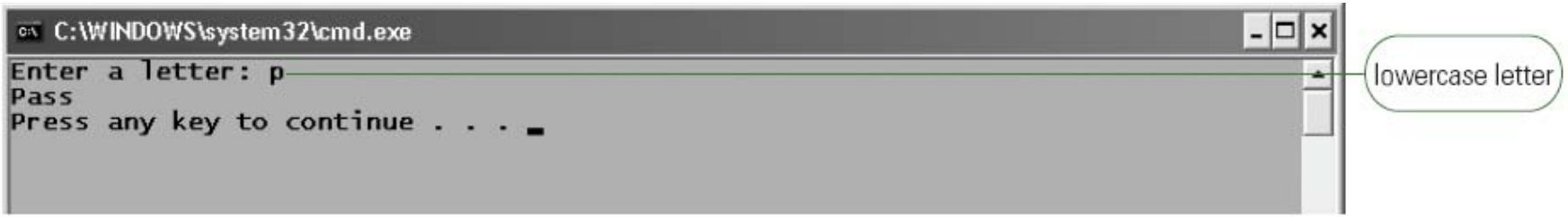


Figure 5-23: Sample run of the Pass/Fail program

Comparing Characters (continued)

Example 1: using the || (Or) operator

```
int main()
{
     //declare variable
     char letter = ' ';

     //enter input item
     cout << "Enter a letter: ";
     cin >> letter;

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

     return 0;
}    //end of main function
```

Example 2: using the && (And) operator

```
int main()
{
     //declare variable
     char letter = ' ';

     //enter input item
     cout << "Enter a letter: ";
     cin >> letter;

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

     return 0;
}    //end of main function
```

Example 3: less efficient solution

```
int main()
{
     //declare variable
     char letter = ' ';
```

Figure 5-24: C++ code showing character comparisons in the `if` statement's *condition*

Comparing Characters (continued)

```
    //enter input item
    cout << "Enter a letter: ";
    cin >> letter;

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

    return 0;
}   //end of main function
```

Example 4: using the `toupper()` function

```
int main()
{
    //declare variable
    char letter = ' ';

    //enter input item
    cout << "Enter a letter: ";
    cin >> letter;

    //display message
    if (toupper(letter) == 'P')
        cout << "Pass" << endl;
    else
        cout << "Fail" << endl;
    //end if

    return 0;
}   //end of main function
```

Figure 5-24: C++ code showing character comparisons in the `if` statement's *condition (Continued)*

Converting a Character to Uppercase or Lowercase

Use the `toupper()` and `tolower()` Functions

Syntax

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

Examples

```
if (toupper(letter) == 'P')
letter = toupper(letter);
if (tolower(letter) == 'p')
```

Figure 5-25: How to use the `toupper()` and `tolower()` functions

Comparing Strings

- String comparisons are case-sensitive
  - “yes”, “YES”, and “Yes” are different
- Before using a string in a comparison, convert it to uppercase or lowercase
  - Use `transform()`

Converting a String to Uppercase or Lowercase

Use the `transform()` Function

Syntax

transform(*string***.begin(), *string*.end(), *string*.begin(), *function*);**

Examples

```
transform(state.begin(), state.end(), state.begin(), tolower)
transform(item.begin(), item.end(), item.begin(), toupper)
```

```
#include
<algorithm>
using
std::transform;
```

Figure 5-26: How to use the `transform()` function

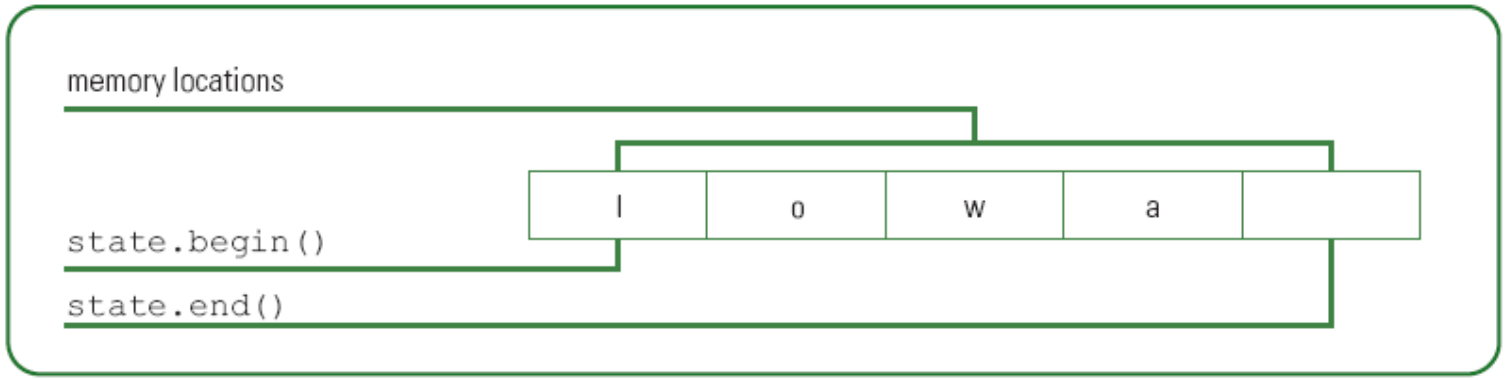


Figure 5-27: Illustration of `state.begin()` and `state.end()`

`transform()` Function Program: Calculating Sales Tax

- Calculate and display the amount of sales tax that a customer owes
- Tax rate is 3% of purchase price
- If purchase is made in Gunter County, tax rate is 3.25%

transform() Function Program: Calculating Sales Tax (continued)

C++ code – Calculating sales tax program

```
int main()
{
     //declare constants
     const double TAXRATE        = .03;
     const double GUNTER_TAXRATE = .0325;

     //declare variables
     double purchase = 0.0;
     string county   = "";
     double tax      = 0.0;

     //enter input items
     cout << "Purchase price: ";
     cin >> purchase;
     cin.ignore(100, '\n');
     cout << "County: ";
     getline(cin, county);

     //convert county name to uppercase
     transform(county.begin(), county.end(),
               county.begin(), toupper);

     //calculate and display sales tax
     if (county == "GUNTER")
          tax = purchase * GUNTER_TAXRATE;
     else
          tax = purchase * TAXRATE;
     //end if

     cout << fixed << setprecision(2);
     cout << "Tax: " << tax << endl;
     //end if

     return 0;
}    //end of main function
```

transform() function converts the county name to uppercase

Figure 5-28: C++ code for the calculating sales tax program

`transform()` Function Program: Calculating Sales Tax (continued)

Figure 5-29: First sample run of the calculating sales tax program

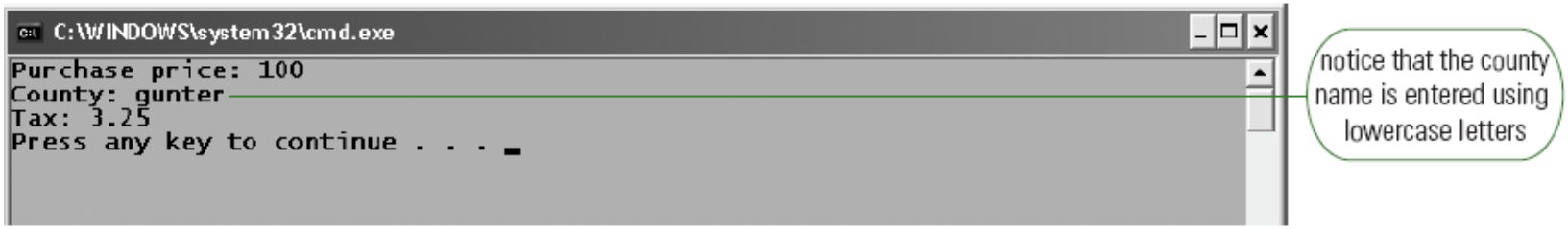


Figure 5-30: Second sample run of the calculating sales tax program

Summary

- The selection structure (decision structure) is one of the three programming structures
  - Forms: `if`, `if/else`, and `switch` (or `case`)
  - Represented by a diamond in a flowchart
  - Use the C++ `if` statement to code both the `if` and `if/else` forms
    - *Condition* can contain variables, constants, functions, and arithmetic, comparison, or logical operators
- Character and string comparisons are case sensitive
  - Use `toupper()`, `tolower()` and `transform()`

Application Lesson: Using the Selection Structure in a C++ Program

- Lab 5.1: Stop and Analyze
- Lab 5.2
  - Create the Willow Springs Health Club program
- Lab 5.3
  - Modify program so that the `if` statement's true path handles the invalid data, and its false path handles the valid data
- Lab 5.4: Desk-Check Lab
- Lab 5.5: Debugging Lab