Chapter 4: Control Structures I
   (Selection)
Objectives

In this chapter, you will:

• Learn about control structures
• Examine relational and logical operators
• Explore how to form and evaluate logical (Boolean) expressions
• Discover how to use the selection control structures if, if...else, and switch in a program
Objectives (cont’d.)

• Learn how to avoid bugs by avoiding partially understood concepts
• Learn to use the `assert` function to terminate a program
Control Structures

• A computer can proceed:
  – In sequence
  – Selectively (branch): making a choice
  – Repetitively (iteratively): looping

• Some statements are executed only if certain conditions are met

• A condition is met if it evaluates to true
Control Structures (cont'd.)

**FIGURE 4-1** Flow of execution
Relational Operators

• A condition is represented by a logical (Boolean) expression that can be true or false

• Relational operators:
  – Allow comparisons
  – Require two operands (binary)
  – Evaluate to true or false
### TABLE 4-1  Relational Operators in C++

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>==</code></td>
<td>equal to</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>not equal to</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>less than</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>less than or equal to</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>greater than</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>greater than or equal to</td>
</tr>
</tbody>
</table>
Relational Operators and Simple Data Types

• You can use the relational operators with all three simple data types:
  – 8 < 15 evaluates to true
  – 6 != 6 evaluates to false
  – 2.5 > 5.8 evaluates to false
  – 5.9 <= 7.5 evaluates to true
Comparing Characters

• Expression with relational operators
  – Depends on machine’s collating sequence
  – ASCII character set

• Logical (Boolean) expressions
  – Expressions such as $4 < 6$ and $'R' > 'T'$
  – Returns an integer value of 1 if the logical expression evaluates to true
  – Returns an integer value of 0 otherwise
Relational Operators and the `string` Type

- Relational operators can be applied to strings
- Strings are compared character by character, starting with the first character
- Comparison continues until either a mismatch is found or all characters are found equal
- If two strings of different lengths are compared and the comparison is equal to the last character of the shorter string
  - The shorter string is less than the larger string
Relational Operators and the string Type (cont'd.)

• Suppose we have the following declarations:

```c++
string str1 = "Hello";
string str2 = "Hi";
string str3 = "Air";
string str4 = "Bill";
string str4 = "Big";
```
### Relational Operators and the `string` Type (cont'd.)

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value /Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str1 &lt; str2</code></td>
<td><code>true</code>&lt;br&gt;<code>str1 = &quot;Hello&quot;</code> and <code>str2 = &quot;Hi&quot;</code>. The first characters of <code>str1</code> and <code>str2</code> are the same, but the second character 'e' of <code>str1</code> is less than the second character 'i' of <code>str2</code>. Therefore, <code>str1 &lt; str2</code> is <code>true</code>.</td>
</tr>
<tr>
<td><code>str1 &gt; &quot;Hen&quot;</code></td>
<td><code>false</code>&lt;br&gt;<code>str1 = &quot;Hello&quot;</code>. The first two characters of <code>str1</code> and &quot;Hen&quot; are the same, but the third character 'l' of <code>str1</code> is less than the third character 'n' of &quot;Hen&quot;. Therefore, <code>str1 &gt; &quot;Hen&quot;</code> is <code>false</code>.</td>
</tr>
<tr>
<td><code>str3 &lt; &quot;An&quot;</code></td>
<td><code>true</code>&lt;br&gt;<code>str3 = &quot;Air&quot;</code>. The first characters of <code>str3</code> and &quot;An&quot; are the same, but the second character 'i' of &quot;Air&quot; is less than the second character 'n' of &quot;An&quot;. Therefore, <code>str3 &lt; &quot;An&quot;</code> is <code>true</code>.</td>
</tr>
<tr>
<td><code>str1 == &quot;hello&quot;</code></td>
<td><code>false</code>&lt;br&gt;<code>str1 = &quot;Hello&quot;</code>. The first character 'H' of <code>str1</code> is less than the first character 'h' of &quot;hello&quot; because the ASCII value of 'H' is 72, and the ASCII value of 'h' is 104. Therefore, <code>str1 == &quot;hello&quot;</code> is <code>false</code>.</td>
</tr>
<tr>
<td><code>str3 &lt;= str4</code></td>
<td><code>true</code>&lt;br&gt;<code>str3 = &quot;Air&quot;</code> and <code>str4 = &quot;Bill&quot;</code>. The first character 'A' of <code>str3</code> is less than the first character 'B' of <code>str4</code>. Therefore, <code>str3 &lt;= str4</code> is <code>true</code>.</td>
</tr>
<tr>
<td><code>str2 &gt; str4</code></td>
<td><code>true</code>&lt;br&gt;<code>str2 = &quot;Hi&quot;</code> and <code>str4 = &quot;Bill&quot;</code>. The first character 'H' of <code>str2</code> is greater than the first character 'B' of <code>str4</code>. Therefore, <code>str2 &gt; str4</code> is <code>true</code>.</td>
</tr>
</tbody>
</table>
Relational Operators and the string Type (cont'd.)

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value/Explanation</th>
</tr>
</thead>
</table>
| \texttt{str4} \texttt{\geq} "Billy" | false  
|                     | \texttt{str4} = "Bill". It has four characters, and "Billy" has five characters. Therefore, \texttt{str4} is the shorter string. All four characters of \texttt{str4} are the same as the corresponding first four characters of "Billy", and "Billy" is the larger string. Therefore, \texttt{str4} \texttt{\geq} "Billy" is false. |
| \texttt{str5} \texttt{\leq} "Bigger" | true  
|                     | \texttt{str5} = "Big". It has three characters, and "Bigger" has six characters. Therefore, \texttt{str5} is the shorter string. All three characters of \texttt{str5} are the same as the corresponding first three characters of "Bigger", and "Bigger" is the larger string. Therefore, \texttt{str5} \texttt{\leq} "Bigger" is true. |
Logical (Boolean) Operators and Logical Expressions

- Logical (Boolean) operators enable you to combine logical expressions.

**TABLE 4-2 Logical (Boolean) Operators in C++**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Logical (Boolean) Operators and Logical Expressions (cont'd.)

**TABLE 4-3  The ! (Not) Operator**

<table>
<thead>
<tr>
<th>Expression</th>
<th>!(Expression)</th>
</tr>
</thead>
<tbody>
<tr>
<td>true (nonzero)</td>
<td>false (0)</td>
</tr>
<tr>
<td>false (0)</td>
<td>true (1)</td>
</tr>
</tbody>
</table>

**EXAMPLE 4-3**

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(&quot;A' &gt; 'B&quot;)</td>
<td>true</td>
<td>Because 'A' &gt; 'B' is false, !(&quot;A' &gt; 'B&quot;) is true.</td>
</tr>
<tr>
<td>!(6 &lt;= 7)</td>
<td>false</td>
<td>Because 6 &lt;= 7 is true, !(6 &lt;= 7) is false.</td>
</tr>
</tbody>
</table>
### Logical (Boolean) Operators and Logical Expressions (cont'd.)

**TABLE 4-4  The && (And) Operator**

<table>
<thead>
<tr>
<th>Expression1</th>
<th>Expression2</th>
<th>Expression1 &amp;&amp; Expression2</th>
</tr>
</thead>
<tbody>
<tr>
<td>true (nonzero)</td>
<td>true (nonzero)</td>
<td>true (1)</td>
</tr>
<tr>
<td>true (nonzero)</td>
<td>false (0)</td>
<td>false (0)</td>
</tr>
<tr>
<td>false (0)</td>
<td>true (nonzero)</td>
<td>false (0)</td>
</tr>
<tr>
<td>false (0)</td>
<td>false (0)</td>
<td>false (0)</td>
</tr>
</tbody>
</table>

**EXAMPLE 4-4**

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(14 &gt;= 5) &amp;&amp; ('A' &lt; 'B')</td>
<td>true</td>
<td>Because (14 &gt;= 5) is true, ('A' &lt; 'B') is true, and true &amp;&amp; true is true, the expression evaluates to true.</td>
</tr>
<tr>
<td>(24 &gt;= 35) &amp;&amp; ('A' &lt; 'B')</td>
<td>false</td>
<td>Because (24 &gt;= 35) is false, ('A' &lt; 'B') is true, and false &amp;&amp; true is false, the expression evaluates to false.</td>
</tr>
</tbody>
</table>
Logical (Boolean) Operators and Logical Expressions (cont'd.)

### TABLE 4-5 The || (Or) Operator

| Expression1               | Expression2       | Expression1 || Expression2 |
|---------------------------|-------------------|--------------|
| true (nonzero)            | true (nonzero)    | true (1)     |
| true (nonzero)            | false (0)         | true (1)     |
| false (0)                 | true (nonzero)    | true (1)     |
| false (0)                 | false (0)         | false (0)    |

### EXAMPLE 4-5

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(14 &gt;= 5)</td>
<td></td>
<td>('A' &gt; 'B')</td>
</tr>
<tr>
<td>(24 &gt;= 35)</td>
<td></td>
<td>('A' &gt; 'B')</td>
</tr>
<tr>
<td>('A' &lt;= 'a')</td>
<td></td>
<td>(7 != 7)</td>
</tr>
</tbody>
</table>
Order of Precedence

• Relational and logical operators are evaluated from left to right
• The associativity is left to right
• Parentheses can override precedence
### Order of Precedence (cont'd.)

#### TABLE 4-6 Precedence of Operators

<table>
<thead>
<tr>
<th>Operators</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>!, +, − (unary operators)</td>
<td>first</td>
</tr>
<tr>
<td>* , /, %</td>
<td>second</td>
</tr>
<tr>
<td>+, −</td>
<td>third</td>
</tr>
<tr>
<td>&lt;, &lt;=, &gt;, &gt;=, &gt;</td>
<td>fourth</td>
</tr>
<tr>
<td>==, ! =</td>
<td>fifth</td>
</tr>
<tr>
<td>&amp; &amp;</td>
<td>sixth</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>= (assignment operator)</td>
<td>last</td>
</tr>
</tbody>
</table>
Order of Precedence (cont'd.)

EXAMPLE 4-6

Suppose you have the following declarations:

```cpp
bool found = true;
int age = 20;
double hours = 45.30;
double overtime = 15.00;
int count = 20;
char ch = 'B';
```
<table>
<thead>
<tr>
<th>Expression</th>
<th>Value / Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>!found</td>
<td>false</td>
</tr>
<tr>
<td>hours &gt; 40.00</td>
<td>true</td>
</tr>
<tr>
<td>!age</td>
<td>false</td>
</tr>
<tr>
<td>!found &amp;&amp; (age &gt;= 18)</td>
<td>false</td>
</tr>
<tr>
<td>!(found &amp;&amp; (age &gt;= 18))</td>
<td>false</td>
</tr>
</tbody>
</table>

Because found is true, !found is false.

Because hours is 45.30 and 45.30 > 40.00 is true, the expression hours > 40.00 evaluates to true.

Age is 20, which is nonzero, so age is true. Therefore, !age is false.

!found is false; age > 18 is 20 > 18 is true. Therefore, !found && (age >= 18) is false && true, which evaluates to false.

Now, found && (age >= 18) is true && true, which evaluates to true. Therefore, !(found && (age >= 18)) is !true, which evaluates to false.
Order of Precedence (cont'd.)

Expression

hours + overTime <= 75.00

Value / Explanation

true

Because hours + overTime is 45.30 + 15.00 = 60.30 and 60.30 <= 75.00 is true, it follows that
hours + overTime <= 75.00 evaluates to true.

(count >= 0) &&
(count <= 100)

true

Now, count is 20. Because 20 >= 0 is true,
count >= 0 is true. Also, 20 <= 100 is true, so
count <= 100 is true. Therefore, (count >=
0) && (count <= 100) is true && true,
which evaluates to true.

('A' <= ch && ch <= 'Z')

true

Here, ch is 'B'. Because 'A' <= 'B' is true,
'A' <= ch evaluates to true. Also, because 'B'
<= 'Z' is true, ch <= 'Z' evaluates to true.
Therefore, ('A' <= ch && ch <= 'Z') is true
&& true, which evaluates to true.
Data Type and Logical (Boolean) Expressions

- Earlier versions of C++ did not provide built-in data types that had Boolean values.
- Logical expressions evaluate to either 1 or 0.
  - The value of a logical expression was stored in a variable of the data type `int`.
- You can use the `int` data type to manipulate logical (Boolean) expressions.
The **bool** Data Type and Logical (Boolean) Expressions

- The data type **bool** has logical (Boolean) values **true** and **false**
- **bool**, **true**, and **false** are reserved words
- The identifier **true** has the value **1**
- The identifier **false** has the value **0**
Selection: `if` and `if...else`

- One-Way Selection
- Two-Way Selection
- Compound (Block of) Statements
- Multiple Selections: Nested `if`
- Comparing `if...else` Statements with a Series of `if` Statements
One-Way Selection

• The syntax of one-way selection is:

\[
\text{if (expression) statement}
\]

• The statement is executed if the value of the expression is \textit{true}

• The statement is bypassed if the value is \textit{false}; program goes to the next statement

• \textit{if} is a reserved word
One-Way Selection (cont'd.)

**FIGURE 4-2** One-way selection
One-Way Selection (cont'd.)

**EXAMPLE 4-7**

```c++
if (score >= 60)
    grade = 'P';
```

In this code, if the expression \((\text{score} \geq 60)\) evaluates to \texttt{true}, the assignment statement, \(\text{grade} = 'P';\), executes. If the expression evaluates to \texttt{false}, the statements (if any) following the \texttt{if} structure execute. For example, if the value of \texttt{score} is 65, the value assigned to the variable \texttt{grade} is \texttt{'P'}. 
One-Way Selection (cont'd.)

EXAMPLE 4-8

The following C++ program finds the absolute value of an integer.

//Program: Absolute value of an integer

#include <iostream>

using namespace std;

int main()
{
    int number, temp;

    cout << "Line 1: Enter an integer: ";       //Line 1
    cin >> number;
    cout << endl;                           //Line 2

    temp = number;                          //Line 4

    if (number < 0)                           //Line 5
        number = -number;

    cout << "Line 7: The absolute value of "  //Line 7
        << temp << " is " << number << endl;

    return 0;
}

Sample Run: In this sample run, the user input is shaded.

Line 1: Enter an integer: -6734
Line 7: The absolute value of -6734 is 6734
One-Way Selection (cont'd.)

EXAMPLE 4-9

Consider the following statement:

```cpp
if score >= 60 //syntax error
grade = 'P';
```

This statement illustrates an incorrect version of an if statement. The parentheses around the logical expression are missing, which is a syntax error.

EXAMPLE 4-10

Consider the following C++ statements:

```cpp
if (score >= 60); //Line 1
grade = 'P'; //Line 2
```

Because there is a semicolon at the end of the expression (see Line 1), the if statement in Line 1 terminates. The action of this if statement is null, and the statement in Line 2 is not part of the if statement in Line 1. Hence, the statement in Line 2 executes regardless of how the if statement evaluates.
Two-Way Selection

• Two-way selection takes the form:

```cpp
if (expression)  
  statement1  
else  
  statement2
```

• If expression is `true`, `statement1` is executed; otherwise, `statement2` is executed
  – `statement1` and `statement2` are any C++ statements

• `else` is a reserved word
Two-Way Selection (cont'd.)

**FIGURE 4-3** Two-way selection
Two-Way Selection (cont'd.)

Example 4-11

Consider the following statements:

```c++
if (hours > 40.0) {
    wages = 40.0 * rate +
           1.5 * rate * (hours - 40.0);
} else {
    wages = hours * rate;
}
```

//Line 1
//Line 2
//Line 3
//Line 4

If the value of the variable `hours` is greater than 40.0, the `wages` include overtime payment. Suppose that `hours` is 50. The expression in the `if` statement, in Line 1, evaluates to `true`, so the statement in Line 2 executes. On the other hand, if `hours` is 30 or any number less than or equal to 40, the expression in the `if` statement, in Line 1, evaluates to `false`. In this case, the program skips the statement in Line 2 and executes the statement in Line 4—that is, the statement following the reserved word `else` executes.
Two-Way Selection (cont'd.)

EXAMPLE 4-12

The following statements show an example of a syntax error.

```c++
if (hours > 40.0);  //Line 1
    wages = 40.0 * rate +
        1.5 * rate * (hours - 40.0);  //Line 2
else
    wages = hours * rate;          //Line 4
```

The semicolon at the end of the if statement (see Line 1) ends the if statement, so the statement in Line 2 separates the else clause from the if statement. That is, else is all by itself. Because there is no stand-alone else statement in C++, this code generates a syntax error. As shown in Example 4-10, in a one-way selection, the semicolon at the end of an if statement is a logical error, whereas as shown in this example, in a two-way selection, it is a syntax error.
Compound (Block of) Statements

- Compound statement (block of statements):

```cpp
{ 
    statement1
    statement2
    ...
    statementn
}
```

- A compound statement is a single statement
if (age > 18)
{
    cout << "Eligible to vote." << endl;
    cout << "No longer a minor." << endl;
}
else
{
    cout << "Not eligible to vote." << endl;
    cout << "Still a minor." << endl;
}
Multiple Selections: Nested if

- **Nesting**: one control statement in another
- An `else` is associated with the most recent `if` that has not been paired with an `else`
Multiple Selections: Nested $\texttt{if}$ (cont'd.)

**Example 4-15**

Suppose that $\texttt{balance}$ and $\texttt{interestRate}$ are variables of type $\texttt{double}$. The following statements determine the $\texttt{interestRate}$ depending on the value of the $\texttt{balance}$.

```cpp
if (balance > 50000.00)                   //Line 1
    interestRate = 0.07;                //Line 2
else                                        //Line 3
    if (balance >= 25000.00)           //Line 4
        interestRate = 0.05;           //Line 5
    else                                //Line 6
        if (balance >= 1000.00)       //Line 7
            interestRate = 0.03;       //Line 8
        else                        //Line 9
            interestRate = 0.00;       //Line 10
```
Multiple Selections: Nested \texttt{if} (cont'd.)

To avoid excessive indentation, the code in Example 4-15 can be rewritten as follows:

```cpp
if (balance > 50000.00) //Line 1
    interestRate = 0.07; //Line 2
else if (balance >= 25000.00) //Line 3
    interestRate = 0.05; //Line 4
else if (balance >= 10000.00) //Line 5
    interestRate = 0.03; //Line 6
else //Line 7
    interestRate = 0.00; //Line 8
```
Multiple Selections: Nested if (cont'd.)

EXAMPLE 4-16

Assume that score is a variable of type int. Based on the value of score, the following code outputs the grade:

```cpp
if (score >= 90)
    cout << "The grade is A." << endl;
else if (score >= 80)
    cout << "The grade is B." << endl;
else if (score >= 70)
    cout << "The grade is C." << endl;
else if (score >= 60)
    cout << "The grade is D." << endl;
else
    cout << "The grade is F." << endl;
```
Comparing `if...else` Statements with a Series of `if` Statements

```cpp
a. if (month == 1)
    cout << "January" << endl;
else if (month == 2)
    cout << "February" << endl;
else if (month == 3)
    cout << "March" << endl;
else if (month == 4)
    cout << "April" << endl;
else if (month == 5)
    cout << "May" << endl;
else if (month == 6)
    cout << "June" << endl;

b. if (month == 1)
    cout << "January" << endl;
if (month == 2)
    cout << "February" << endl;
if (month == 3)
    cout << "March" << endl;
if (month == 4)
    cout << "April" << endl;
if (month == 5)
    cout << "May" << endl;
if (month == 6)
    cout << "June" << endl;
```
Short-Circuit Evaluation

- **Short-circuit evaluation**: evaluation of a logical expression stops as soon as the value of the expression is known

- Example:

  ```
  (age >= 21) || (x == 5) //Line 1
  (grade == 'A') && (x >= 7) //Line 2
  ```
Comparing Floating-Point Numbers for Equality: A Precaution

• Comparison of floating-point numbers for equality may not behave as you would expect
  – Example:
    • $1.0 == 3.0/7.0 + 2.0/7.0 + 2.0/7.0$ evaluates to false
    • Why? $3.0/7.0 + 2.0/7.0 + 2.0/7.0 = 0.99999999999999989$

• Solution: use a tolerance value
  – Example: $\text{fabs}(x - y) < 0.000001$
Associativity of Relational Operators: A Precaution

```cpp
#include <iostream>

using namespace std;

int main()
{
    int num;

    cout << "Enter an integer: ";
    cin >> num;
    cout << endl;

    if (0 <= num <= 10)
        cout << num << " is within 0 and 10." << endl;
    else
        cout << num << " is not within 0 and 10." << endl;

    return 0;
}
```
Associativity of Relational Operators: A Precaution (cont’d.)

- `num = 5`

<table>
<thead>
<tr>
<th>0 &lt;= num &lt;= 10</th>
<th>= 0 &lt;= 5 &lt;= 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>= (0 &lt;= 5) &lt;= 10</td>
<td>(Because relational operators are evaluated from left to right)</td>
</tr>
<tr>
<td>= 1 &lt;= 10</td>
<td>(Because 0 &lt;= 5 is true, 0 &lt;= 5 evaluates to 1)</td>
</tr>
<tr>
<td>= 1 (true)</td>
<td></td>
</tr>
</tbody>
</table>

- `num = 20`

<table>
<thead>
<tr>
<th>0 &lt;= num &lt;= 10</th>
<th>= 0 &lt;= 20 &lt;= 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>= (0 &lt;= 20) &lt;= 10</td>
<td>(Because relational operators are evaluated from left to right)</td>
</tr>
<tr>
<td>= 1 &lt;= 10</td>
<td>(Because 0 &lt;= 20 is true, 0 &lt;= 20 evaluates to 1)</td>
</tr>
<tr>
<td>= 1 (true)</td>
<td></td>
</tr>
</tbody>
</table>
Avoiding Bugs by Avoiding Partially Understood Concepts and Techniques

• Must use concepts and techniques correctly;
  – Otherwise solution will be either incorrect or deficient

• If you do not understand a concept or technique completely
  – Don’t use it
  – Save yourself an enormous amount of debugging time
Input Failure and the `if` Statement

- If input stream enters a fail state
  - All subsequent input statements associated with that stream are ignored
  - Program continues to execute
  - May produce erroneous results
- Can use `if` statements to check status of input stream
- If stream enters the fail state, include instructions that stop program execution
Confusion Between the Equality (==) and Assignment (=) Operators

• C++ allows you to use any expression that can be evaluated to either true or false as an expression in the if statement:

```cpp
if (x = 5)
    cout << "The value is five." << endl;
```

• The appearance of = in place of == resembles a silent killer
  – It is not a syntax error
  – It is a logical error
Conditional Operator (?:)

- Conditional operator (?:) takes three arguments
  - Ternary operator
- Syntax for using the conditional operator:
  expression1 ? expression2 : expression3
- If `expression1` is true, the result of the conditional expression is `expression2`
  - Otherwise, the result is `expression3`
Program Style and Form (Revisited): Indentation

• If your program is properly indented
  – Spot and fix errors quickly
  – Show the natural grouping of statements

• Insert a blank line between statements that are naturally separate

• Two commonly used styles for placing braces
  – On a line by themselves
  – Or left brace is placed after the expression, and the right brace is on a line by itself
Using Pseudocode to Develop, Test, and Debug a Program

• Pseudocode, or just pseudo
  – Informal mixture of C++ and ordinary language
  – Helps you quickly develop the correct structure of the program and avoid making common errors

• Use a wide range of values in a walk-through to evaluate the program
switch Structures

- **switch structure**: alternate to if-else
- **switch (integral)** expression is evaluated first
- Value of the expression determines which corresponding action is taken
- Expression is sometimes called the selector

```
switch (expression)
{
    case value1:
        statements1
        break;
    case value2:
        statements2
        break;
    
    case valuen:
        statementsn
        break;
    default:
        statements
}
```
switch Structures (cont'd.)

![Diagram of switch statement]

**Figure 4-4** switch statement
switch Structures (cont'd.)

- One or more statements may follow a case label
- Braces are not needed to turn multiple statements into a single compound statement
- The \texttt{break} statement may or may not appear after each statement
- \texttt{switch}, \texttt{case}, \texttt{break}, \texttt{and} \texttt{default} are reserved words
switch Structures (cont'd.)

**EXAMPLE 4-21**

Consider the following statements, in which grade is a variable of type `char`.

```
switch (grade) {
  case 'A':
    cout << "The grade point is 4.0.";
    break;
  case 'B':
    cout << "The grade point is 3.0.";
    break;
  case 'C':
    cout << "The grade point is 2.0.";
    break;
  case 'D':
    cout << "The grade point is 1.0.";
    break;
  case 'F':
    cout << "The grade point is 0.0.";
    break;
  default:
    cout << "The grade is invalid.";
}
```

In this example, the expression in the `switch` statement is a variable identifier. The variable `grade` is of type `char`, which is an integral type. The possible values of `grade` are 'A', 'B', 'C', 'D', and 'F'. Each `case` label specifies a different action to take, depending on the value of `grade`. If the value of `grade` is 'A', the output is:

The grade point is 4.0.
Avoiding Bugs by Avoiding Partially Understood Concepts and Techniques: Revisited

• To output results correctly
  – The switch structure must include a break statement after each cout statement
Terminating a Program with the `assert` Function

- Certain types of errors that are very difficult to catch can occur in a program
  - Example: division by zero can be difficult to catch using any of the programming techniques examined so far

- The predefined function, `assert`, is useful in stopping program execution when certain elusive errors occur
The `assert` Function (cont'd.)

- Syntax:

```c
assert(expression);
```

- `expression` is any logical expression
- If `expression` evaluates to true, the next statement executes
- If `expression` evaluates to false, the program terminates and indicates where in the program the error occurred
- To use `assert`, include `cassert` header file
The `assert` Function (cont'd.)

- `assert` is useful for enforcing programming constraints during program development
- After developing and testing a program, remove or disable assert statements
- The preprocessor directive `#define NDEBUG` must be placed before the directive `#include <cassert>` to disable the assert statement
Programming Example: Cable Company Billing

• This programming example calculates a customer’s bill for a local cable company
• There are two types of customers:
  – Residential
  – Business
• Two rates for calculating a cable bill:
  – One for residential customers
  – One for business customers
Programming Example: Rates

• For residential customer:
  – Bill processing fee: $4.50
  – Basic service fee: $20.50
  – Premium channel: $7.50 per channel

• For business customer:
  – Bill processing fee: $15.00
  – Basic service fee: $75.00 for first 10 connections/$5.00 for each additional one
  – Premium channel cost: $50.00 per channel for any number of connections
Programming Example: Requirements

- Ask user for account number and customer code
- Assume \( R \) or \( r \) stands for residential customer and \( B \) or \( b \) stands for business customer
Programming Example: Input and Output

• Input:
  – Customer account number
  – Customer code
  – Number of premium channels
  – For business customers, number of basic service connections

• Output:
  – Customer’s account number
  – Billing amount
Programming Example: Program Analysis

• Purpose: calculate and print billing amount

• Calculating billing amount requires:
  – Customer for whom the billing amount is calculated (residential or business)
  – Number of premium channels to which the customer subscribes

• For a business customer, you need:
  – Number of basic service connections
  – Number of premium channels
Programming Example: Program Analysis (cont'd.)

- Data needed to calculate the bill, such as bill processing fees and the cost of a premium channel, are known quantities.
- The program should print the billing amount to two decimal places.
Programming Example: Algorithm Design

- Set precision to two decimal places
- Prompt user for account number and customer type
- If customer type is R or r
  - Prompt user for number of premium channels
  - Compute and print the bill
- If customer type is B or b
  - Prompt user for number of basic service connections and number of premium channels
  - Compute and print the bill
Programming Example: Variables and Named Constants

```c++
int accountNumber;  //variable to store the customer's account number
char customerType;  //variable to store the customer code
int numOfPremChannels;  //variable to store the number of premium channels to which the customer subscribes
int numOfBasicServConn;  //variable to store the number of basic service connections to which the customer subscribes
double amountDue;  //variable to store the billing amount

//Named constants - residential customers
const double RES_BILL_PROC_FEES = 4.50;
const double RES_BASIC_SERV_COST = 20.50;
const double RES_COST_PREM_CHANNEL = 7.50;

//Named constants - business customers
const double BUS_BILL_PROC_FEES = 15.00;
const double BUS_BASIC_SERV_COST = 75.00;
const double BUS_BASIC_CONN_COST = 5.00;
const double BUS_COST_PREM_CHANNEL = 50.00;
```
Programming Example: Formulas

Billing for residential customers:

\[
\text{amountDue} = \text{RES\_BILL\_PROC\_FEES} + \\
\text{RES\_BASIC\_SERV\_COST} + \text{numOfPremChannels} \times \\
\text{RES\_COST\_PREM\_CHANNEL};
\]
Programming Example: Formulas (cont'd.)

Billing for business customers:

```c++
if (numOfBasicServConn <= 10)
    amountDue = BUS_BILL_PROC_FEES +
                BUS_BASIC_SERV_COST
              + numOfPremChannels *
                BUS_COST_PREM_CHANNEL;
else
    amountDue = BUS_BILL_PROC_FEES +
                BUS_BASIC_SERV_COST
              + (numOfBasicServConn - 10)
                * BUS_BASIC_CONN_COST
              + numOfPremChannels *
                BUS_COST_PREM_CHANNEL;
```
Programming Example: Main Algorithm

1. Output floating-point numbers in fixed decimal with decimal point and trailing zeros
   – Output floating-point numbers with two decimal places and set the precision to two decimal places

2. Prompt user to enter account number

3. Get customer account number

4. Prompt user to enter customer code

5. Get customer code
Programming Example: Main Algorithm (cont'd.)

6. If the customer code is \( r \) or \( R \),
   - Prompt user to enter number of premium channels
   - Get the number of premium channels
   - Calculate the billing amount
   - Print account number and billing amount
7. If customer code is \( b \) or \( B \),
   - Prompt user to enter number of basic service connections
   - Get number of basic service connections
   - Prompt user to enter number of premium channels
   - Get number of premium channels
   - Calculate billing amount
   - Print account number and billing amount
8. If customer code is other than $r$, $R$, $b$, or $B$, output an error message
Summary

• Control structures alter normal control flow
• Most common control structures are selection and repetition
• Relational operators: ==, <, <=, >, >=, !=
• Logical expressions evaluate to \(1\) (true) or \(0\) (false)
• Logical operators: ! (not), && (and), || (or)
Summary (cont'd.)

• Two selection structures: one-way selection and two-way selection
• The expression in an if or if...else structure is usually a logical expression
• No stand-alone else statement in C++
  – Every else has a related if
• A sequence of statements enclosed between braces, { and }, is called a compound statement or block of statements
Summary (cont'd.)

• Using assignment in place of the equality operator creates a semantic error
• `switch` structure handles multiway selection
• `break` statement ends `switch` statement
• Use `assert` to terminate a program if certain conditions are not met