Chapter 4: Selection Structures
Objectives

• In this chapter, you will learn about:
  – Selection criteria
  – The `if-else` statement
  – Nested `if` statements
  – The `switch` statement
  – Program testing
  – Common programming errors
Selection Criteria

• **if-else** statement: Implements a decision structure for two alternatives

Syntax:

```cpp
if (condition)
    statement executed if condition is true;
else
    statement executed if condition is false;
```
• The condition is evaluated to its numerical value:
  – A non-zero value is considered to be true
  – A zero value is considered to be false
• The `else` portion is optional
  – Executed only if the condition is false
• The condition may be any valid C++ expression
• **Relational expression**: Compares two operands or expressions using **relational operators**

<table>
<thead>
<tr>
<th>Relational Operator</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;</code></td>
<td>Less than</td>
<td>age &lt; 30</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>Greater than</td>
<td>height &gt; 6.2</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>Less than or equal to</td>
<td>taxable &lt;= 20000</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>Greater than or equal to</td>
<td>temp &gt;= 98.6</td>
</tr>
<tr>
<td><code>==</code></td>
<td>Equal to</td>
<td>grade == 100</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>Not equal to</td>
<td>number != 250</td>
</tr>
</tbody>
</table>

**Table 4.1 C++’s Relational Operators**
Relational Operators (continued)

- Relational expressions are evaluated to a numerical value of 1 or 0 only:
  - If the value is 1, the expression is true
  - If the value is 0, the expression is false
- `char` values are automatically coerced to `int` values for comparison purposes
- Strings are compared on a character by character basis
  - The string with the first lower character is considered smaller
## Relational Operators (continued)

### Examples of string comparisons

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Interpretation</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Hello&quot; &gt; &quot;Good-bye&quot;</td>
<td>1</td>
<td>true</td>
<td>The first H in Hello is greater than the first G in Good-bye.</td>
</tr>
<tr>
<td>&quot;SMITH&quot; &gt; &quot;JONES&quot;</td>
<td>1</td>
<td>true</td>
<td>The first S in SMITH is greater than the first J in JONES.</td>
</tr>
<tr>
<td>&quot;123&quot; &gt; &quot;1227&quot;</td>
<td>1</td>
<td>true</td>
<td>The third character in 123, the 3, is greater than the third character in 1227, the 2.</td>
</tr>
<tr>
<td>&quot;Behop&quot; &gt; &quot;Beehive&quot;</td>
<td>1</td>
<td>true</td>
<td>The third character in Behop, the h, is greater than the third character in Beehive, the second e.</td>
</tr>
</tbody>
</table>
Logical Operators

- **AND (&&)**: Condition is true only if both expressions are true
- **OR (||)**: Condition is true if either one or both of the expressions is true
- **NOT (!)**: Changes an expression to its opposite state; true becomes false, false becomes true
## Logical Operators (continued)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>! unary - ++ --</td>
<td>Right to left</td>
</tr>
<tr>
<td>* / %</td>
<td>Left to right</td>
</tr>
<tr>
<td>+ -</td>
<td>Left to right</td>
</tr>
<tr>
<td>&lt; &lt;= &gt; &gt;=</td>
<td>Left to right</td>
</tr>
<tr>
<td>== !=</td>
<td>Left to right</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Left to right</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>= += -= *= /=</td>
<td>Right to left</td>
</tr>
</tbody>
</table>

**Table 4.2 Operator Precedence and Associativity**
A Numerical Accuracy Problem

- Comparing single and double precision values for equality (==) can lead to errors because values are stored in binary
- Instead, test that the absolute value of the difference is within an acceptable range
  - Example:
    
    \[
    \text{abs(operandOne} - \text{operandTwo) < 0.000001}
    \]
The *if-else* Statement

- *if-else* performs instructions based on the result of a comparison
- Place statements on separate lines for readability
- Syntax:

```cpp
if (expression)   // no semicolon here
    statement1;
else               // no semicolon here
    statement2;
```
The if-else Statement (cont’d)

Figure 4.2
The if-else flowchart
The if-else Statement (continued)

Program 4.1

```cpp
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
    double radius;

    cout << "Please type in the radius: ";
    cin >> radius;

    if (radius < 0.0)
        cout << "A negative radius is invalid" << endl;
    else
        cout << "The area of this circle is " << 3.1416 * pow(radius, 2) << endl;

    return 0;
}
```
Compound Statements

• **Compound statement:** A sequence of single statements contained between braces
  – Creates a block of statements
  – A block of statements can be used anywhere that a single statement is legal
  – Any variable declared within a block is usable only within that block

• **Scope:** The area within a program where a variable can be used
  – A variable’s scope is based on where the variable is declared
```cpp
{     // start of outer block
    int a = 25;
    int b = 17;

    cout << "The value of a is " << a
         << " and b is " << b << endl;

    {     // start of inner block
        double a = 46.25;

        int c = 10;
        cout << "a is now " << a
             << " b is now " << b
             << " and c is " << c << endl;
    }     // end of inner block

    cout << "a is now " << a
         << " and b is " << b << endl;
}     // end of outer block
```
• One-way selection: An \texttt{if} statement without the optional \texttt{else} portion

\textbf{Figure 4.3} A one-way selection \texttt{if} statement
Problems Associated with the if-else Statement

• Common problems with if-else statements:
  – Misunderstanding what an expression is
  – Using the assignment operator (=) instead of the relational operator (==)
Nested if Statements

• *if-else* statement can contain any valid C++ statement, including another *if-else*

• Nested *if* statement: an *if-else* statement completely contained within another *if-else*

• Use braces to block code, especially when inner *if* statement does not have its own *else*
Nested if Statements (continued)

Figure 4.4a
Nested within the if part
The if-else Chain

- **if-else** chain: A nested if statement occurring in the else clause of the outer if-else
- If any condition is true, the corresponding statement is executed and the chain terminates
- Final else is only executed if no conditions were true
  - Serves as a catch-all case
- if-else chain provides one selection from many possible alternatives
Figure 4.4b
Nested within the else part
The if-else Chain (continued)

- General form of an if-else chain

```c++
if (expression_1)
    statement1;
else if (expression_2)
    statement2;
else if (expression_3)
    statement3;
    ...
    ...
else if (expression_n)
    statementn;
else
    last_statement;
```
### The switch Statement

- **switch statement**: Provides for one selection from many alternatives
- **switch** keyword starts the statement
  - Is followed by the expression to be evaluated
- **case** keyword identifies a value to be compared to the switch expression
  - When a match is found, statements in this **case** block are executed
- **All further cases after a match is found are executed unless a **break** statement is found**
• `default` case is executed if no other case value matches were found
• `default` case is optional
A Case Study: Solving Quadratic Equations

• **Data validation**: Use defensive programming techniques to validate user input
  – Includes code to check for improper data before an attempt is made to process it further

• **Solving quadratic equations**: Use the software development procedure to solve for the roots of a quadratic equation
• Theory: A comprehensive set of test runs would test all combinations of input and computations, and would reveal all errors.

• Reality: There are too many combinations to test for any program except a very simple one.

• Example:
  – One program with 10 modules, each with five if statements, always called in the same order.
  – There are $2^5$ paths through each module, and more than $2^{50}$ paths through the program!
· Conclusion: there is no error-free program, only one in which no errors have recently been encountered
Common Programming Errors

- Using the assignment operator (\(=\)) instead of the relational operator (\(==\)) for an equality test
- Placing a semicolon immediately after the condition
- Assuming a structural problem with an `if-else` causes the error instead of focusing on the data value being tested
- Using nested `if` statements without braces to define the structure
Summary

- Relational expressions, or conditions, are used to compare operands
- If the relation expression is true, its value is 1; if false, its value is 0
- Use logical operators `&&` (AND), `||` (OR), and `!` (NOT) to construct complex conditions
- `if-else` allows selection between two alternatives
Summary (continued)

- An `if` expression that evaluates to 0 is false; if non-zero, it is true
- `if` statements can be nested
- Chained `if` statement provides a multiway selection
- Compound statement: contains any number of individual statements enclosed in braces
Summary (continued)

- **switch statement**: Provides a multiway selection
- **switch expression**: Evaluated and compared to each `case` value
  - If a match is found, execution begins at that case’s statements and continues unless a `break` is encountered