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
  – By calling a function

• Two most common control structures:
  – Selection
  – Repetition
Control Structures (cont’d.)

a. Sequence
b. Selection
c. Repetition

FIGURE 4-1 Flow of execution
Relational Operators

- Conditional statements: only executed if certain conditions are met
- Condition: represented by a logical (Boolean) expression that evaluates to a logical (Boolean) value of 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>==</td>
<td>equal to</td>
</tr>
<tr>
<td>!=</td>
<td>not equal to</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
</tbody>
</table>
Relational Operators and Simple Data Types

• Relational operators can be used with all three simple data types:
  – $8 < 15$ evaluates to $true$
  – $6 \not= 6$ evaluates to $false$
  – $2.5 > 5.8$ evaluates to $false$
  – $5.9 \leq 7.5$ evaluates to $true$
Comparing Characters

• Expression of char values with relational operators
  – Result 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:

```cpp
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>
</tbody>
</table>
Relational Operators and the `string` Type (cont’d.)

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>str1 == &quot;hello&quot;</code></td>
<td><code>false</code></td>
<td><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></td>
<td><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></td>
<td><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>
<tr>
<td>Expression</td>
<td>Value/Explanation</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>str4 &gt;= &quot;Billy&quot;</td>
<td><strong>false</strong>&lt;br&gt;str4 = &quot;Bill&quot;. It has four characters, and &quot;Billy&quot; has five characters. Therefore, str4 is the shorter string. All four characters of str4 are the same as the corresponding first four characters of &quot;Billy&quot;, and &quot;Billy&quot; is the larger string. Therefore, str4 &gt;= &quot;Billy&quot; is <strong>false</strong>.</td>
<td></td>
</tr>
<tr>
<td>str5 &lt;= &quot;Bigger&quot;</td>
<td><strong>true</strong>&lt;br&gt;str5 = &quot;Big&quot;. It has three characters, and &quot;Bigger&quot; has six characters. Therefore, str5 is the shorter string. All three characters of str5 are the same as the corresponding first three characters of &quot;Bigger&quot;, and &quot;Bigger&quot; is the larger string. Therefore, str5 &lt;= &quot;Bigger&quot; is <strong>true</strong>.</td>
<td></td>
</tr>
</tbody>
</table>
Logical (Boolean) Operators and Logical Expressions

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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>!</code></td>
<td>not</td>
</tr>
<tr>
<td><code>&amp;&amp;</code></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 &quot;A' &gt; 'B&quot; 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><code>!, +, −</code> (unary operators)</td>
<td>first</td>
</tr>
<tr>
<td><code>*, /, %</code></td>
<td>second</td>
</tr>
<tr>
<td><code>+</code>, <code>−</code></td>
<td>third</td>
</tr>
<tr>
<td><code>&lt;</code>, <code>&lt;=</code>, <code>&gt;=</code>, <code>&gt;</code></td>
<td>fourth</td>
</tr>
<tr>
<td><code>==</code>, <code>!=</code></td>
<td>fifth</td>
</tr>
<tr>
<td><code>&amp;&amp;</code></td>
<td>sixth</td>
</tr>
<tr>
<td>`</td>
<td></td>
</tr>
<tr>
<td><code>=</code> (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';
```
Consider the following expressions:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value / Explanation</th>
</tr>
</thead>
</table>
| !found                        | *false*  
  Because *found* is *true*, *!found* is *false*.                                                                                                  |
| hours > 40.00                  | *true*  
  Because *hours* is 45.30 and 45.30 > 40.00 is *true*, the expression *hours* > 40.00 evaluates to *true*.                                              |
| !age                          | *false*  
  *age* is 20, which is nonzero, so *age* is *true*.  
  Therefore, *!age* is *false*.                                                                                                                          |
| !found && (age >= 18)          | *false*  
  *found* is *false*; *age* > 18 is 20 > 18 is *true*.  
  Therefore, *!found && (age >= 18)* is *false && true*, which evaluates to *false*.  
| !(found && (age >= 18))        | *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          Value / Explanation

hours + overtime <= 75.00          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.
The `int` 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
  – Logical expression value was stored in a variable of the data type `int`
• 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
• `if` and `if...else` statements can be used to create:
  – One-way selection
  – Two-way selection
  – Multiple selections
One-Way Selection

• One-way selection syntax:

  ```
  if (expression)
  statement
  ```

• Statement is executed if the value of the expression is true
• Statement is bypassed if the value is false; program goes to the next statement
• Expression is called a decision maker
One-Way Selection (cont’d.)

![Flowchart](image)

**FIGURE 4-2** One-way selection
Two-Way Selection

- Two-way selection syntax:

```c++
if (expression)
    statement1
else
    statement2
```

- If expression is true, statement1 is executed; otherwise, statement2 is executed
  - statement1 and statement2 are any C++ statements
Compound (Block of) Statements

• Compound statement (block of statements):

```cpp
{ statement_1
  statement_2
  ...
  ...
  statement_n
}
```

• A compound statement functions like 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 is located within another
• An *else* is associated with the most recent *if* that has not been paired with an *else*
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

```c++
a. if (month == 1)
    cout << "January" << endl;  //Line 1
    cout << "February" << endl;  //Line 2
else if (month == 2)
    cout << "March" << endl;    //Line 3
else if (month == 3)
    cout << "April" << endl;   //Line 4
else if (month == 4)
    cout << "May" << endl;   //Line 5
else if (month == 5)
    cout << "June" << endl;  //Line 6
else if (month == 6)
    cout << "July" << endl;  //Line 7
```

C++ Programming: Program Design Including Data Structures, Sixth Edition
Comparing \texttt{if...else Statements} with \texttt{if Statements} (cont’d.)

b. \texttt{if} (month == 1) 
   \hspace{1em} cout << "January" << endl;
\texttt{if} (month == 2) 
   \hspace{1em} cout << "February" << endl;
\texttt{if} (month == 3) 
   \hspace{1em} cout << "March" << endl;
\texttt{if} (month == 4) 
   \hspace{1em} cout << "April" << endl;
\texttt{if} (month == 5) 
   \hspace{1em} cout << "May" << endl;
\texttt{if} (month == 6) 
   \hspace{1em} 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: $\text{if } \text{fabs}(x - y) < 0.000001$
#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 (\leq) num (\leq) 10</th>
<th>= 0 (\leq) 5 (\leq) 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>= (0 (\leq) 5) (\leq) 10</td>
<td>(Because relational operators are evaluated from left to right)</td>
</tr>
<tr>
<td>= 1 (\leq) 10</td>
<td>(Because 0 (\leq) 5 is true, 0 (\leq) 5 evaluates to 1)</td>
</tr>
<tr>
<td>= 1 (true)</td>
<td></td>
</tr>
</tbody>
</table>

- **num = 20**

<table>
<thead>
<tr>
<th>0 (\leq) num (\leq) 10</th>
<th>= 0 (\leq) 20 (\leq) 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>= (0 (\leq) 20) (\leq) 10</td>
<td>(Because relational operators are evaluated from left to right)</td>
</tr>
<tr>
<td>= 1 (\leq) 10</td>
<td>(Because 0 (\leq) 20 is true, 0 (\leq) 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 (?:)**
  - Ternary operator: takes 3 arguments
- **Syntax for the conditional operator:**
  \[ \text{expression}_1 \ ? \ \text{expression}_2 \ : \ \text{expression}_3 \]
- **If \text{expression}_1 \ is \ true, the result of the conditional expression is \text{expression}_2**
  - Otherwise, the result is \text{expression}_3
- **Example:** \( \text{max} = (a \ >= \ b) \ ? \ a \ : \ b; \)
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 Structures (cont’d.)

![Diagram of switch statement structure](image)
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
• When a case value is matched, all statements after it execute until a break is encountered
• The break statement may or may not appear after each statement
• switch, case, break, and default are reserved words
switch Structures (cont’d.)

EXAMPLE 4-21

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

```c++
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: Revisited

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

- Certain types of errors are very difficult to catch
  - Example: division by zero
- `assert` function: useful in stopping program execution when certain elusive errors occur
The assert Function (cont’d.)

• Syntax:
  
  ```
  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.
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)
• 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