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.)

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**

Relational Operators (cont’d.)

Relational Operators and Simple Data Types

- Relational operators can be used 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 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:
  
  ```
  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>str1 == str2</td>
<td>True. The first character of str1 and str2 are the same, but the second character ‘H’ of str1 is less than the second character ‘H’ of str2. Therefore, str1 &lt; str2 is true.</td>
</tr>
<tr>
<td>str1 == &quot;Max&quot;</td>
<td>True. str1 == &quot;Max&quot;. The first two characters of str1 and &quot;Max&quot; are the same, but the third character ‘M’ of str1 is less than the second character ‘M’ of &quot;Max&quot;. Therefore, str1 &lt; str2 is true.</td>
</tr>
<tr>
<td>str2 == &quot;Max&quot;</td>
<td>False. The first character of str2 and &quot;Max&quot; are the same, but the second character ‘I’ of str2 is less than the second character ‘M’ of &quot;Max&quot;. Therefore, str2 &lt; str2 is true.</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>!</td>
<td>not</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Example 4-3

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

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.)**

**Example 4.6**

Suppose you have the following declarations:

```c++
bool found = true;
int age = 25;
double hours = 45.10;
```

and the following code:

```c++
found = (hours > 40.00) && (age > 25.00);
```

This expression simplifies as follows:

1. `hours > 40.00` evaluates to `true`.
2. `age > 25.00` evaluates to `true`.
3. The `&&` operator requires both operands to be `true` for the overall expression to be `true`.
4. Therefore, the expression simplifies 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

Selection: if and if...else

- `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:
  ```cpp
  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.)

- Figure 4.2: One-way selection

expression → true → statement
false → statement
Two-Way Selection

- Two-way selection syntax:

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

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

Two-Way Selection (cont'd.)

![Two-way selection diagram]

Compound (Block of) Statements

- **Compound statement (block of statements):**

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

- A compound statement functions like a single statement.

Compound (Block of) Statements (cont’d.)

```cpp
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

Comparing if...else Statements with a Series of if Statements

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

Comparing if...else Statements with if Statements (cont’d.)

b. if (month == 1) cout << "January" << endl; //Line 1
if (month == 2) cout << "February" << endl; //Line 2
if (month == 3) cout << "March" << endl; //Line 3
if (month == 4) cout << "April" << endl; //Line 4
if (month == 5) cout << "May" << endl; //Line 5
if (month == 6) cout << "June" << endl; //Line 6

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;
```
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: \( \text{fabs}(x - y) < 0.000001 \)
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:
  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:
  expression1 ? expression2 : expression3
- If expression1 is true, the result of the conditional expression is expression2
  - Otherwise, the result is expression3
- Example: max = (a >= b) ? a : b;
Program Style and Form (Revisited): Indentation

• A properly indented program:
  – Helps you spot and fix errors quickly
  – Shows 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 Structures (cont’d.)

• 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

![Diagram of 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
- 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

**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

**Avoiding Bugs: Revisited**

- To output results correctly
  - Consider whether the `switch` structure must include a `break` statement after each `cout` statement
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)

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