3.1 Control Structures

- **Control structure**: A logical design that controls order in which set of statements execute
- **Sequence structure**: A set of statements that execute in the order they appear
- **Decision structure**: Specific action(s) performed only if a condition exists
  - Also known as selection structure
3.1 Control Structures

• **Flowchart**: A diagram that shows a program's logical flow of execution.

![Flowchart Image]

• **Pseudocode**: an informal language that has no syntax rules and is not meant to be compiled or executed. You don’t have to worry about syntax errors while writing pseudocode, you can focus your attention on the program’s logic.

```
Display “Enter the number of hours the employee worked.”
Input hours
Display “Enter the employee’s hourly pay rate.”
Input payRate
grossPay = hours * payRate
Display grossPay
```
3.2 Writing a Decision Structure with the `if` Statement

• In a decision structure’s simplest form, a specific action is performed only if a certain condition exists. If the condition does not exist, the action is not performed.

• In a flowchart, diamond represents true/false condition that must be tested
3.2 Writing a Decision Structure with the if Statement

- Actions can be *conditionally executed*
  - Performed only when a condition is true
- Single alternative decision structure: provides only one alternative path of execution
  - If condition is not true, exit the structure

C++ syntax:

```cpp
if (expression) {
  statement;
  statement;
  etc;
}
```
3.2 Writing a Decision Structure with the if Statement

• First line starts with the keyword `if` followed by a Boolean expression
  • The expression can be true or false
  • When the `if` statement executes, the expression is tested, and if it is true the statements inside the braces are executed. otherwise, those statements are skipped

```c
if (expression)
{
    statement;
    statement;
    etc;
}
```

• If you are writing an `if` statement that has only one conditionally executed statement, you do not have to enclose the conditionally executed statement inside braces.

```c
if (expression)
    statement;
```
3.2 Writing a Decision Structure with the if Statement

- **Boolean expression**: expression tested by if statement to determine if it is true or false
  - Example: \( a > b \)
    - **true** if \( a \) is greater than \( b \); **false** otherwise

- **Relational operator**: determines whether a specific relationship exists between two values
  - Example: greater than (>)

### Table 3-1: Relational operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>Equal to</td>
</tr>
<tr>
<td>!=</td>
<td>Not equal to</td>
</tr>
</tbody>
</table>

Copyright © 2014 Pearson Education, Inc. Publishing as Pearson Addison-Wesley
3.2 Writing a Decision Structure with the `if` Statement

### Table 3-2 Boolean expressions using relational operators

<table>
<thead>
<tr>
<th>Expression</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x &gt; y</code></td>
<td>Is <code>x</code> greater than <code>y</code>?</td>
</tr>
<tr>
<td><code>x &lt; y</code></td>
<td>Is <code>x</code> less than <code>y</code>?</td>
</tr>
<tr>
<td><code>x &gt;= y</code></td>
<td>Is <code>x</code> greater than or equal to <code>y</code>?</td>
</tr>
<tr>
<td><code>x &lt;= y</code></td>
<td>Is <code>x</code> less than or equal to <code>y</code>?</td>
</tr>
<tr>
<td><code>x == y</code></td>
<td>Is <code>x</code> equal to <code>y</code>?</td>
</tr>
<tr>
<td><code>x != y</code></td>
<td>Is <code>x</code> not equal to <code>y</code>?</td>
</tr>
</tbody>
</table>

Example:

```c
if (sales > 50000)
{
    bonus = 500;
}
```

Copyright © 2014 Pearson Education, Inc. Publishing as Pearson Addison-Wesley
3.2 Writing a Decision Structure with the *if* Statement

• Example:

```cpp
if (sales > 50000)
{
    bonus = 500.0;
    commissionRate = 0.12;
    cout << "You met your sales quota!" << endl;
}
```

3.3 The *if-else* Statement

• Dual alternative decision structure: two possible paths of execution
  - One is taken if the Boolean expression is true, and the other if the expression is false
  - Syntax:
    ```cpp
    if (expression)
    {
        statement;
        statement;
    }
    else
    {
        statement;
        statement;
    }
    ```
3.3 The if-else Statement

If the expression is true, this block of statements is executed.

If the expression is false, this block of statements is executed.

Then, control jumps here, to the statement following the if-else statement.
3.3 The if-else Statement

• If either set of conditionally executed statements contains only one statement, the braces are not required.

```java
if (expression)
    statement;
else
    statement;
```

3.4 Nested Decision Structures and the if-else-if Statement

• A decision structure can be nested inside another decision structure
  – Commonly needed in programs
  – Example:
    • Determine if someone qualifies for a loan, they must meet two conditions:
      – Must earn at least $30,000/year
      – Must have been employed for at least two years
    • Check first condition, and if it is true, check second condition
3.4 Nested Decision Structures and the \texttt{if-else-if} Statement

\textit{From Program 3-3}

```c
25    // Determine whether the customer qualifies.
26    if (salary >= MIN_SALARY)
27        if (yearsOnJob >= MIN_YEARS)
28            cout << "You qualify for the loan.\" << endl;
29        else
30            cout << "You must have been on your current job for at least two years to qualify.\" << endl;
31    
32        else
33            cout << "You must earn at least \$" << MIN_SALARY
34            " to qualify.\" << endl;
35    
36    else
37        return 0;
38    
39    }
40    return 0;
```

Copyright © 2014 Pearson Education, Inc. Publishing as Pearson Addision-Wesley
3.4 Nested Decision Structures and the *if-else-if* Statement

- **The if-else-if Statement:**
  - special version of a decision structure
  - Makes logic of nested decision structures simpler to write
```cpp
if (BooleanExpression_1)
{
    statement;
    statement;
    etc.
}
else if (BooleanExpression_2)
{
    statement;
    statement;
    etc.
}

Insert as many else if clauses as necessary.
else
{
    statement;
    statement;
    etc.
}
```

Program 3-5 (Grader2.cpp)

```cpp
1 // This program gets a numeric test score from the
2 // user and displays the corresponding letter grade.
3 #include <iostream>
4 using namespace std;
5
6 int main()
7 {
8    int testScore;
9   10    // Get the test score.
11    cout << "Enter your test score: ";
12    cin >> testScore;
13   14    // Determine the letter grade.
15    if (testScore >= 90)
16    cout << "Your grade is A." << endl;
17    else if (testScore >= 80)
18    cout << "Your grade is B." << endl;
19    else if (testScore >= 70)
20    cout << "Your grade is C." << endl;
21    else if (testScore >= 60)
22    cout << "Your grade is D." << endl;
23    else
24    cout << "Your grade is F." << endl;
25    return 0;
26 }
```

Program Output (with Input Shown in Bold)
Enter your test score: 78 [Enter]
Your grade is C.

Program Output (with Input Shown in Bold)
Enter your test score: 84 [Enter]
Your grade is B.
3.4 Nested Decision Structures and the if-else-if Statement

- Alignment used with if-else-if statement:
  - if and else clauses are all aligned
  - Conditionally executed blocks are consistently indented
- if-else-if statement is never required, but logic easier to follow
  - Can be accomplished by nested if-else
    - Code can become complex, and indentation can cause problematic long lines

3.5 Logical Operators

- Operators that can be used to create complex Boolean expressions
  - && is the logical AND operator
  - || is the logical OR operator
  - ! is the logical NOT operator
3.5 Logical Operators

- && operator and | | operator: binary operators, connect two Boolean expressions into a compound Boolean expression
- ! operator: unary operator, reverses the truth of its Boolean operand

The && Operator
- Takes two Boolean expressions as operands
  - Creates compound Boolean expression that is true only when both sub expressions are true
  - Can be used to simplify nested decision structures
- Truth table for the && operator

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value of the Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>false and false</td>
<td>false</td>
</tr>
<tr>
<td>false and true</td>
<td>false</td>
</tr>
<tr>
<td>true and false</td>
<td>false</td>
</tr>
<tr>
<td>true and true</td>
<td>true</td>
</tr>
</tbody>
</table>
3.5 Logical Operators

• **The | | Operator**
  - Takes two Boolean expressions as operands
    - Creates compound Boolean expression that is true when either sub expressions are true
    - Can be used to simplify nested decision structures
  - Truth table for the | | operator

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value of the Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>false and false</td>
<td>false</td>
</tr>
<tr>
<td>false and true</td>
<td>true</td>
</tr>
<tr>
<td>true and false</td>
<td>true</td>
</tr>
<tr>
<td>true and true</td>
<td>true</td>
</tr>
</tbody>
</table>

• **Short circuit evaluation**: deciding the value of a compound Boolean expression after evaluating only one sub expression
  - Performed by the | | and && operators
    - For | | operator: If left operand is true, compound expression is true. Otherwise, evaluate right operand
    - For && operator: If left operand is false, compound expression is false. Otherwise, evaluate right operand
3.5 Logical Operators

• The ! Operator
  – Takes one Boolean expressions as operand and reverses its logical value
  • Sometimes it may be necessary to place parentheses around an expression to clarify to what you are applying the not operator
  – Truth table for the \texttt{not} operator

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value of the Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

• Checking Numeric Ranges with Logical Operators
  – To determine whether a numeric value is within a specific range of values, use \&\&
    – Example: \texttt{x >= 10 \&\& x <= 20}
  – To determine whether a numeric value is outside of a specific range of values, use ||
    – Example: \texttt{x < 10 || x > 20}
3.6 The `switch` Statement

- The `switch` statement lets the value of a variable or an expression determine which path of execution the program will take.
3.6 The `switch` Statement

```cpp
switch (month) {
    case 1:
        cout << "January" << endl;
        break;
    case 2:
        cout << "February" << endl;
        break;
    case 3:
        cout << "March" << endl;
        break;
    default:
        cout << "Error: Invalid month" << endl;
}
```

3.7 `bool` Variables

- A `bool` variable references one of two values, `true` or `false`
  - Represented by `bool` data type
- Commonly used as flags
  - **Flag**: variable that signals when some condition exists in a program
    - Flag set to `false` → condition does not exist
    - Flag set to `true` → condition exists
3.8 Comparing Strings

- Strings can be compared using the `==` and `!=` operators
- String comparisons are case sensitive
- Strings can be compared using `>`, `<`, `>=`, and `<=`
  - Compared character by character based on the ASCII values for each character
  - If shorter word is substring of longer word, longer word is greater than shorter word