Conditions, logical expressions, and selection

Introduction to control structures
Flow of control

• In a program, statements execute in a particular order

• By default, statements are executed *sequentially*:
  – One after another, from top to bottom
  – One at a time
  – Exactly once

• This sequential flow of control can be altered through the use of *control structures*
Control structures

• There are two general types of control structures:
  – Selection (aka branching) structures: cause flow of control to take, or not take, a particular direction
  – Repetition (aka iteration, looping) structures: cause a set of statements to execute several times

• Both of these type of structures depend on the evaluation of logical expressions
Logical expressions

• Logical expressions evaluate to true or false; the result of a logical expression is always a value of data type `boolean`

• Most logical expressions use relational and logical operators

• In Java the form of a simple logical expression is:
  
  Operand1 operator Operand2

  – The operands can be simple or compound expressions of any type
  – The operator is a relational operator
Relational operators

• The relational operators in Java include:
  
  <     :     is less than
  <=    :     is less than or equal to
  ==    :     equals
  >     :     is greater than
  >=    :     is greater than or equal to
  !=    :     does not equal
Logical expression examples

- Suppose you had the following declarations:
  \[ \text{int } x = 3, y = 7; \]
- Then the following expressions would have the values indicated:
  \[ x > y \quad // \quad \text{(false)} \]
  \[ y >= x \quad // \quad \text{(true)} \]
  \[ x != y \quad // \quad \text{(true)} \]
  \[ (x > y) == \text{true} \quad // \quad \text{(false)} \]
  \[ ((y >= x) == (x != y)) \quad // \quad \text{true} \]
  \[ x = y \quad // \quad \text{value is 7; not a logical expression} \]
Logical operators

• Three operators in Java can be used to form compound logical expressions (expressions that combine simple logical, or relational expressions)
• They are:
  & & - logical and
  | | - logical or
  ! - logical not
Logical operators

- Logical and (&&) combines two expressions; if both sub-expressions are true, then the compound expression is true – otherwise, the compound expression is false.
- Logical or also combines two expressions; the compound expression is true if one or both sub-expressions is true, false otherwise.
- Logical not reverses the truth value of an expression; if the original expression was true, not makes it false, and vice versa.
Truth table

- Graphical display of relationships between truth values of propositions
- Shows all possible values of propositions, or combinations of propositions
- Suppose p represents an expression; then the truth table for \( \neg p \) is as show below:

<table>
<thead>
<tr>
<th>p</th>
<th>( \neg p )</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>
Truth table for p && q

Suppose p and q represent two logical sub-expressions; then the compound expression p && q has the following truth table:

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>p &amp;&amp; q</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
Truth table for $p \parallel q$

Suppose $p$ and $q$ represent two logical sub-expressions; then the compound expression $p \parallel q$ has the following truth table:

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$p \parallel q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
### Partial listing of operator precedence in Java

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>NOT</td>
<td>Right</td>
</tr>
<tr>
<td>*, / , %</td>
<td>Multiplication, Division, Modulus</td>
<td>Left</td>
</tr>
<tr>
<td>+ , -</td>
<td>Addition, Subtraction</td>
<td>Left</td>
</tr>
<tr>
<td>&lt;</td>
<td>Less than</td>
<td>Left</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than or equal to</td>
<td>Left</td>
</tr>
<tr>
<td>&gt;</td>
<td>Greater than</td>
<td>Left</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than or equal to</td>
<td>Left</td>
</tr>
<tr>
<td>==</td>
<td>Is equal to</td>
<td>Left</td>
</tr>
<tr>
<td>!=</td>
<td>Is not equal to</td>
<td>Left</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>AND</td>
<td>Left</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>Assignment</td>
<td>Right</td>
</tr>
</tbody>
</table>
int age;
boolean isSenior, hasFever;
double temperature;

age = 20;
temperature = 102.0;
isSenior = (age >= 55);
hasFever = (temperature > 98.6);

<table>
<thead>
<tr>
<th>EXPRESSION</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>isSenior &amp;&amp; hasFever</td>
<td>false</td>
</tr>
<tr>
<td>isSenior</td>
<td></td>
</tr>
<tr>
<td>! isSenior</td>
<td>true</td>
</tr>
<tr>
<td>! hasFever</td>
<td>false</td>
</tr>
</tbody>
</table>
What is the value?

```java
int age, height;
age = 25;
height = 70;
```

<table>
<thead>
<tr>
<th>EXPRESSION</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(age &lt; 10)</td>
<td>?</td>
</tr>
</tbody>
</table>
What is the value?

int age, height;

age = 25;
height = 70;

<table>
<thead>
<tr>
<th>EXPRESSION</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(height &gt; 60)</td>
<td>?</td>
</tr>
</tbody>
</table>
“Short-Circuit” Evaluation

- Java uses short circuit evaluation of logical expressions

- This means logical expressions are evaluated left to right and evaluation stops as soon as the final truth value can be determined
Short-Circuit Example

```java
int age, height;

age = 25;
height = 70;
```

**EXPRESSION**

(age > 50) && (height > 60)

false

Evaluation can stop now because result of && is only true when both sides are true. It is already determined that the entire expression will be false.
More Short-Circuiting

```c
int age, height;
age = 25;
height = 70;
```

**EXPRESSION**

```c
(height > 60) || (age > 40)
```

true

Evaluation can stop now because result of `||` is true if one side is true. It is already determined that the entire expression will be true.
What happens?

```c
int age, weight;
age = 25;
weight = 145;
```

**EXPRESSION**

(weight < 180) && (age >= 20)

true

Must still be evaluated because truth value of entire expression is not yet known. Why? Result of && is only true if both sides are true.
What happens?

```c
int age, height;
age = 25;
height = 70;
```

**EXPRESSION**

```
! (height > 60) || (age > 50)
```

Does this part need to be evaluated?

(true = yes, false = no)
Equivalent expressions

- Each of the following expressions means "temperature is less than or equal to 75 or humidity is less than 70%"

\[
\begin{align*}
(temp & \leq 75) \lor (humid < 0.7) \\
(temp & = 75) \lor (temp < 75) \lor (humid < 0.7) \\
!(temp > 75) \lor (humid < 0.7)
\end{align*}
\]
Are they equivalent?

Which of the following expressions means “age is over 21 and age is less than 60?”

A. $21 < \text{age} < 60$
B. $60 > \text{age} > 21$
C. $(\text{age} > 21) \&\& (\text{age} < 60)$
D. All of the above
Are they equivalent?

- True or false: both of the following expressions mean “age is 21 or 22”

  age == 21 || 22
  age == 21 || age == 22