Rational (Boolean) Expressions

Relational operators

Logical operators

Order of precedence
Rational (Boolean) Expressions
Expressions

- any combination of variables and constants that can be evaluated to yield a result
- typically involve *operators*
- *Examples:*
  - 5
  - x
  - x + y
  - num++
  - a = 3 + j
Relational Expressions

- compare operands
- used in decision making
- evaluate to 1 (true) or 0 (false)

<table>
<thead>
<tr>
<th>Operand</th>
<th>Relational Operator</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>price</td>
<td>&lt;</td>
<td>34.98</td>
</tr>
</tbody>
</table>
Relational Operators
Relational Operators

less than $<$
greater than $>$
less than or equal to $\leq$
greater than or equal to $\geq$
Equal to $==$  
Not Equal to $!=$
a - b < 0 is equivalent to (a - b) < 0
Expressions such as $4 < 6$ is an example of a logical (Boolean) expression.

When C++ evaluates a logical expression, it returns an integer value of 1 if the logical expression evaluates to true; it returns an integer value of 0 otherwise.

```cpp
int x;
x = 4 < 6;
cout << x;  // 1
```

In C++, any nonzero value is treated as true, and a zero value is treated as false.

```cpp
cout << (4 < 6);  // 1
Must be between brackets
```
# Relational Operators

Can use any relational operator

<table>
<thead>
<tr>
<th>Expression</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>int &lt; int</td>
<td>3 &lt; 4</td>
</tr>
<tr>
<td>float &gt; float</td>
<td>6.2 &gt; 4.2</td>
</tr>
<tr>
<td>char &gt; char</td>
<td>‘a’ &lt; ‘A’</td>
</tr>
<tr>
<td>int &lt; float</td>
<td>2 &lt; 3.1</td>
</tr>
<tr>
<td>int &lt; char</td>
<td>66 &gt; ’A’</td>
</tr>
<tr>
<td>variable &lt; arithmetic operation</td>
<td>x &gt; 3+y</td>
</tr>
<tr>
<td>arithmetic operation &lt; arithmetic</td>
<td>x+7 &lt; y--</td>
</tr>
</tbody>
</table>
### Relational Operators and Simple Data Types

<table>
<thead>
<tr>
<th>Expression</th>
<th>Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 &lt; 15</td>
<td>8 is less than 15</td>
<td>true</td>
</tr>
<tr>
<td>6 != 6</td>
<td>6 is not equal to 6</td>
<td>false</td>
</tr>
<tr>
<td>2.5 &gt; 5.8</td>
<td>2.5 is greater than 5.8</td>
<td>false</td>
</tr>
<tr>
<td>5.9 &lt;= 7.5</td>
<td>5.9 is less than or equal to 7.5</td>
<td>true</td>
</tr>
</tbody>
</table>
Relational Operators and equality Operations

Examples

Valid

a < 3
a > b
-1.1 >= (2.2 * x + 3.3)
k != -2
y == 2 * z - 5

Not Valid

a =< b // out of order
a < = b // space not allowed
a = = b // space not allowed
a = b // assignment statement
a = = b - 1 // space not allowed
y =! z // this is equivalent to y = (!z)
Equality Operators Examples

1. void main()
   {
   int x=7;
   int y=5;
   y=! x;
   cout<<y;
   getch();
   }

   0

2. void main()
   {
   int x=7;
   int y=5;
   y!= x;
   cout<<y;
   getch();
   }

   5

3. void main()
   {
   int x=7;
   int y=5;
   cout<<(y!= x);
   getch();
   }

   1

4. void main()
   {
   int x=7;
   int y;
   cout<<(!(y!= x));
   getch();
   }

   0
Logical Operators and logical expressions
Logical Operators and logical expressions

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

- In C++, there are three logical (Boolean) operators:
  - Negation: `!`
  - Logical and: `&&`
  - Logical or: `||`

- The operator `!` is unary, so it has only one operand.
- The operators `&&` and `||` are binary operators.
Logical Operators and logical expressions

- ! Variable
- ! (Logical exp)
- ! value
- variable && variable
- variable && exp
- exp && exp
- The same in ||

- ! x
- ! (4>2)
- ! 6
- x && y
- x && (5==3)
- (3<5) && (6>33)
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**

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<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>
## 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

#### Expression

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(14 &gt;= 5) &amp;&amp; (-1 &lt; 2)</td>
<td>true</td>
<td>Because (14 &gt;= 5) is true, (-1 &lt; 2) is true, and true &amp;&amp; true is true, the expression evaluates to true.</td>
</tr>
<tr>
<td>(24 &gt;= 35) &amp;&amp; (-1 &lt; 2)</td>
<td>false</td>
<td>Because (24 &gt;= 35) is false, (-1 &lt; 2) is true, and false &amp;&amp; true is false, the expression evaluates to false.</td>
</tr>
</tbody>
</table>
# 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**

**Expression**

\[(14 \geq 5) \lor (7 > 13)\]

**Value**

true

**Explanation**

Because \((14 \geq 5)\) is true, \((7 > 13)\) is false, and true \(\lor\) false is true, the expression evaluates to true.

**Expression**

\[(24 \geq 35) \lor (7 > 13)\]

**Value**

false

**Explanation**

Because \((24 \geq 35)\) is false, \((7 > 13)\) is false, and false \(\lor\) false is false, the expression evaluates to false.
Examples

Valid
a && b
a || b && c
!(a < b) && c
3 && (-2 * a + 7)

Not Valid
a && // one operand missing
a | b // extra space not allowed
Order of Precedence
Order of Precedence

Consider the logical expression:

\[ 11 > 5 \text{ } || \text{ } 6 < 15 \text{ } && \text{ } 7 \geq 8 \]

- This logical expression will yield different results if \( || \) is evaluated first or \( && \) is evaluated first.
- If \( || \) is evaluated first, this logical expression evaluates to 0 (false).
- If \( && \) is evaluated first, this logical expression evaluates to 1 (true).
# Precedence of Operators

<table>
<thead>
<tr>
<th>Operators</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>!, +, – (unary operators)</td>
<td>first</td>
</tr>
<tr>
<td>*, /, %</td>
<td>second</td>
</tr>
<tr>
<td>+, –</td>
<td>third</td>
</tr>
<tr>
<td>&lt;, &lt;=, &gt;=, &gt;</td>
<td>fourth</td>
</tr>
<tr>
<td>==, !=</td>
<td>fifth</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>sixth</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>= (assignment operator)</td>
<td>last</td>
</tr>
</tbody>
</table>
Logical Operators: Example

int a = 0, b = 3, c = 1, d = 4;

```
a && b || !c || d
```

2 1 3 4

F F 1 T
## Logical Operators

<table>
<thead>
<tr>
<th>Expression</th>
<th>Expression Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(a == b)</td>
<td>a != b</td>
</tr>
<tr>
<td>!(a == b</td>
<td></td>
</tr>
<tr>
<td>!(a == b &amp;&amp; c &gt; d)</td>
<td>a != b</td>
</tr>
</tbody>
</table>
True or False

- Remember False evaluates to Zero
- True is any non zero (but when talking about relation expressions and logical expressions True evaluates specifically to one.

(7 && 8==8) → (true && true) → 1
Example

```c++
void main()
{
    int num = 1, a = 5, b = 8, n = 20;
    float x = 5.2, y = 3.4;
    cout<<(x > 4.0)<<endl;
    cout<<!num<<endl;
    cout<<(x + y <= 20.5)<<endl;
    cout<<((n >= 0) && (n <= 100))<<endl;
    getch();
}
```

Output:

```
1
0
1
1
```
You can insert parentheses into an expression to clarify its meaning.

The expression

\[ 11 > 5 \; \text{||} \; 6 < 15 \; \text{&&} \; 7 \geq 8 \]

is equivalent to

\[ 11 > 5 \; \text{||} \; (6 < 15 \; \text{&&} \; 7 \geq 8) \]

This logical expression evaluates to 1 (true).
Example

Evaluate the following expression:

$$(17 < 4 \times 3 + 5) \, || \, (8 \times 2 \, == \, 4 \times 4) \, && \, !(3 + 3 \, == \, 6)$$

$$= (17 < 12 + 5) \, || \, (16 \, == \, 16) \, && \, !(6 \, == \, 6)$$
$$= (17 < 17) \, || \, true \, && \, !(true)$$
$$= false \, || \, true \, && \, false$$
$$= false \, || \, false$$
$$= false$$

When its printed on the screen as an output it displays

0
Example

Consider the following expressions:

1. \((5 \geq 3) \text{ || } (x == 5)\)
2. \((2 == 3) \&\& (x >= 7)\)

- In statement 1, because \((5 \geq 3)\) is true and the logical operator used in the expression is \(||\), the expression evaluates to true. The computer does not evaluate \((x == 5)\).
- In statement 2, because \((2 == 3)\) is false and the logical operator used in the expression is \(&\&\), the expression evaluates to false. The computer does not evaluate \((x >= 7)\).
Logical (Boolean) Assignments

The int Data Type and Logical (Boolean) Expressions

Since logical expressions are evaluated to either 1 or 0, the value of a logical expression can be stored in a variable of the type int. That is, logical (Boolean) expressions were manipulated with the help of int data type.

Example:

```cpp
int legalAge, age;
Cin>>age;

legalAge = (age >= 21);
```

assigns the value 1 to legalAge if the value of age is greater than or equal to 21. The statement assigns the value 0 if the value of age is less than 21.
The Boolean Data type

A Boolean type is an integral type whose variable can have only two values: \texttt{false} and \texttt{true}

These value are stored as integers \texttt{0} and \texttt{1}

\begin{align*}
\text{bool identifier;} & \quad \text{OR} \quad \text{bool identifier= value;} \\
\text{bool identifier= false;} & \quad \text{OR} \quad \text{bool identifier= true;} \\
\end{align*}

\begin{align*}
\text{bool x;} & \quad \text{bool x=true;} & \quad \text{bool x=0;} & \quad \text{bool x=9;} \\
x= !(5 != 2); & \quad \text{cout««x;} & \quad \text{cout««x;} & \quad \text{cout««x;} \\
\text{cout««x;} & \quad \text{0} & \quad \text{1} & \quad \text{0} & \quad \text{1}
\end{align*}
Notes:

■ The expression

\[
0 \leq num \leq 10
\]

Syntax error

■ The correct way to write this expression in C++ is

\[
0 \leq num \text{ && } num \leq 10
\]
Review Question

Assume $a=5$, $b=2$, $c=4$, $d=6$, and $e=3$. Determine the value of each of the following expressions:

- $a > b$
- $a != b$
- $d \% b == c \% b$
- $a * c != d * b$
- $a \% b * c$
Common Programming Errors

- not declaring all variables
- storing data of one type in a variable of a different type.
- using a variable before assigning it a value
- in integer division $4/5 = 0$