Relational and Logical Operators

Topics

- Relational Operators and Expressions
- The if Statement
- The if-else Statement
- Nesting of if-else Statements
- Logical Operators and Expressions
- Truth Tables

Reading

- Sections 2.6, 4.10, 4.11

Relational Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>less than</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
</tr>
<tr>
<td>==</td>
<td>is equal to</td>
</tr>
<tr>
<td>!=</td>
<td>is not equal to</td>
</tr>
</tbody>
</table>

Relational expressions evaluate to the integer values 1 (true) or 0 (false).

All of these operators are called binary operators because they take two expressions as operands.

Practice with Relational Expressions

```c
int a = 1, b = 2, c = 3;

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a &lt; c</td>
<td></td>
<td>a + b &gt;= c</td>
<td></td>
</tr>
<tr>
<td>b &lt;= c</td>
<td></td>
<td>a + b == c</td>
<td></td>
</tr>
<tr>
<td>c &lt;= a</td>
<td></td>
<td>a != b</td>
<td></td>
</tr>
<tr>
<td>a &gt; b</td>
<td></td>
<td>a + b != c</td>
<td></td>
</tr>
<tr>
<td>b &gt;= c</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Arithmetic Expressions: True or False

- **Arithmetic expressions** evaluate to numeric values.
- An arithmetic expression that has a value of zero is false.
- An arithmetic expression that has a value other than zero is true.

Practice with Arithmetic Expressions

```plaintext
int a = 1, b = 2, c = 3;
float x = 3.33, y = 6.66;
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>Numeric Value</th>
<th>True/False</th>
</tr>
</thead>
<tbody>
<tr>
<td>a + b</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b - 2 * a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c - b - a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c - a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y - x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>y - 2 * x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Review: Structured Programming

- All programs can be written in terms of only three control structures
  - The *sequence* structure
    - Unless otherwise directed, the statements are executed in the order in which they are written.
  - The *selection* structure
    - Used to choose among alternative courses of action.
  - The *repetition* structure
    - Allows an action to be repeated while some condition remains true.
Selection: the **if** statement

```c
if ( condition )
{
    statement(s) /* body of the if statement */
}
```

The braces are not required if the body contains only a single statement. However, they are a good idea and are required by the 104 C Coding Standards.

### Examples

```c
if ( age >= 18 )
{
    printf("Vote!\n") ;
}
```

```c
if ( value == 0 )
{
    printf("The value you entered was zero.\n") ;
    printf("Please try again.\n") ;
}
```

### Good Programming Practice

- Always place braces around the body of an if statement.
- Advantages:
  - Easier to read
  - Will not forget to add the braces if you go back and add a second statement to the body
  - Less likely to make a semantic error
- Indent the body of the if statement 3 to 5 spaces -- be consistent!
Selection: the **if-else** statement

```c
if ( condition )
{
    statement(s) /* the if clause */
}
else
{
    statement(s) /* the else clause */
}
```

**Example**

```c
if ( age >= 18 )
{
    printf("Vote!\n") ;
}
else
{
    printf("Maybe next time!\n") ;
}
```

**Example**

```c
if ( value == 0 )
{
    printf("The value you entered was zero.\n") ;
    printf("Please try again.\n") ;
}
else
{
    printf("Value = %d.\n", value) ;
}
```
Good Programming Practice

- Always place braces around the bodies of the if and else clauses of an if-else statement.
- Advantages:
  - Easier to read
  - Will not forget to add the braces if you go back and add a second statement to the clause
  - Less likely to make a semantic error
- Indent the bodies of the if and else clauses 3 to 5 spaces -- be consistent!

Nesting of if-else Statements

```c
if ( condition1 )
{
    statement(s)
}
else if ( condition2 )
{
    statement(s)
} else /* more else clauses may be here */
else /* the default case */
```

Example

```c
if ( value == 0 )
{
    printf ("The value you entered was zero.\n") ;
}
else if ( value < 0 )
{
    printf ("%d is negative.\n", value) ;
}
else
{
    printf ("%d is positive.\n", value) ;
}
```
Gotcha! “==” Versus “==”

```c
int a = 2;
if (a == 1) /* semantic (logic) error! */
  { printf("a is one\n"); }
else if (a == 2)
  { printf("a is two\n"); }
else
  { printf("a is %d\n", a); }
```

Gotcha (con’t)

- The statement if (a == 1) is syntactically correct, so no error message will be produced. (Some compilers will produce a warning.) However, a semantic (logic) error will occur.
- An assignment expression has a value – the value being assigned. In this case the value being assigned is 1, which is true.
- If the value being assigned was 0, then the expression would evaluate to 0, which is false.
- This is a VERY common error. So, if your if-else structure always executes the same, look for this typographical error.

Logical Operators

- So far we have seen only simple conditions.
  
  ```c
  if ( count > 10 ) . . .
  ```
- Sometimes we need to test multiple conditions in order to make a decision.
- Logical operators are used for combining simple conditions to make complex conditions.
  
  ```c
  && is AND if ( x > 5 && y < 6 )
  || is OR if ( z == 0 || x > 10 )
  ! is NOT if (! (bob > 42) )
  ```
Example Use of `&&`

```c
if (age < 1 && gender == 'm')
{
    printf("Infant boy\n");
}
```

Truth Table for `&&`

<table>
<thead>
<tr>
<th>Exp1</th>
<th>Exp2</th>
<th>Exp1 &amp; Exp2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>nonzero</td>
<td>0</td>
</tr>
<tr>
<td>nonzero</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>nonzero</td>
<td>nonzero</td>
<td>1</td>
</tr>
</tbody>
</table>

Exp1 && Exp2 && … && Expn will evaluate to 1 (true) only if ALL subconditions are true.

Example Use of `||`

```c
if (grade == 'D' || grade == 'F')
{
    printf("See you next semester!\n");
}
```
Truth Table for `||`

| Exp₁ | Exp₂ | Exp₁ || Exp₂ |
|------|------|--------|
| 0    | 0    | 0      |
| 0    | nonzero | 1      |
| nonzero | 0    | 1      |
| nonzero | nonzero | 1     |

Exp₁ && Exp₂ && ... && Expₙ will evaluate to 1 (true) if only ONE subcondition is true.

Example Use of `!!`

```c
if (! (x == 2)) /* same as (x != 2) */
{
    printf("x is not equal to 2.\n") ;
}
```

Truth Table for `!`

<table>
<thead>
<tr>
<th>Expression</th>
<th>! Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>nonzero</td>
<td>0</td>
</tr>
</tbody>
</table>
Operator Precedence and Associativity

<table>
<thead>
<tr>
<th>Precedence</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>left to right/inside-out</td>
</tr>
<tr>
<td>++  -- (unary)  - (unary) (type)</td>
<td>right to left</td>
</tr>
<tr>
<td>*  /  %</td>
<td>left to right</td>
</tr>
<tr>
<td>+ (addition) - (subtraction)</td>
<td>left to right</td>
</tr>
<tr>
<td>&lt;  &lt;=  &gt;  &gt;=</td>
<td>left to right</td>
</tr>
<tr>
<td>==  !=</td>
<td>left to right</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>left to right</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>+= -= *= /= %=</td>
<td>right to left</td>
</tr>
<tr>
<td>(comma)</td>
<td></td>
</tr>
</tbody>
</table>

Some Practice Expressions

```
int a = 1, b = 0, c = 7;

<table>
<thead>
<tr>
<th>Expression</th>
<th>Numeric Value</th>
<th>True/False</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>a + b</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>a &amp;&amp; b</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
<td>b</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>!c</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>!!c</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>a &amp;&amp; !b</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>a &lt; b &amp;&amp; b &lt; c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a &gt;= b &amp;&amp; b &lt; c</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a &gt;= b &amp;&amp; b &gt; c</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

More Practice

Given
```
int a = 5, b = 7, c = 17;
```
evaluate each expression as True or False.

1. c / b == 2
2. c % b <= a % b
3. b + c / a == c - a
4. (b < c) && (c == 7)
5. (c + 1 - b == 0) || (b = 5)