Relational Operators and Expressions

- Relational operators evaluate to true or false.
- All of these operators are called binary operators because they take two expressions as operands.

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 Relational Expressions

<table>
<thead>
<tr>
<th>Expression</th>
<th>true/false</th>
<th>Expression</th>
<th>true/false</th>
</tr>
</thead>
<tbody>
<tr>
<td>a &lt; c</td>
<td>true</td>
<td>a + b &gt;= c</td>
<td>true</td>
</tr>
<tr>
<td>b &lt;= c</td>
<td>true</td>
<td>a + b == c</td>
<td>true</td>
</tr>
<tr>
<td>c &lt;= a</td>
<td>true</td>
<td>a != b</td>
<td>true</td>
</tr>
<tr>
<td>a &gt; b</td>
<td>true</td>
<td>a + b != c</td>
<td>true</td>
</tr>
<tr>
<td>b &gt;= c</td>
<td>true</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Practice with Arithmetic Expressions

<table>
<thead>
<tr>
<th>Expression</th>
<th>Numeric Value</th>
<th>True/False</th>
</tr>
</thead>
<tbody>
<tr>
<td>a + b</td>
<td>3</td>
<td>true</td>
</tr>
<tr>
<td>b - 2 * a</td>
<td>-1</td>
<td>false</td>
</tr>
<tr>
<td>c - b - a</td>
<td>1</td>
<td>true</td>
</tr>
<tr>
<td>c - a</td>
<td>-1</td>
<td>false</td>
</tr>
<tr>
<td>y - x</td>
<td>3.33 - 6.66</td>
<td>false</td>
</tr>
<tr>
<td>y - 2 * x</td>
<td>3.33 - 13.32</td>
<td>false</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 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) {
    alert("Go Vote!");
}
if(value == 0) {
    alert("You entered zero.");
}
```

Alert Screenshot

![Alert Screenshot]

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 2 to 3 spaces -- be consistent!

Selection: the if-else statement

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

- Note that there is no condition for the else.
Example

```javascript
if(age >= 18) {
    alert("Go Vote!");
} else {
    alert("Maybe next time!");
}
```

Another Example

```javascript
if(value == 0) {
    alert("You entered zero.");
} else {
    alert("Value = " + 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 2 to 3 spaces -- be consistent!

Nesting of if-else Statements

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

Another Example

```javascript
if(value == 0) {
    alert("You entered zero.");
} else if(value < 0) {
    alert(value + " is negative.");
} else {
    alert(value + " is positive.");
}
```

Gotcha! = versus ==

```javascript
var a = 2;
if(a = 1) /* semantic (logic) error */
    { alert("a is one");
    } else if(a == 2)
    { alert("a is two");
    } else
    { alert("a is " + a); }
```
Gotcha! = versus ==

- The statement `if (a == 1)` is syntactically correct, so no error message will be produced. 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.

Multiple Selection with if

```
if (day == 0) {
    alert("Sunday");
} else if (day == 1) {
    alert("Monday");
} else if (day == 2) {
    alert("Tuesday");
} else if (day == 3) {
    alert("Wednesday");
} else if (day == 4) {
    alert("Thursday");
} else if (day == 5) {
    alert("Friday");
} else if (day == 6) {
    alert("Saturday");
} else {
    alert("Error - invalid day.");
}
```

This if-else structure is more efficient than the corresponding if structure. Why?

Multiple Selection with if-else

```
if (day == 0) {
    alert("Sunday");
} else if (day == 1) {
    alert("Monday");
} else if (day == 2) {
    alert("Tuesday");
} else if (day == 3) {
    alert("Wednesday");
} else if (day == 4) {
    alert("Thursday");
} else if (day == 5) {
    alert("Friday");
} else if (day == 6) {
    alert("Saturday");
} else {
    alert("Error - invalid day.");
}
```

The switch Multiple-Selection Structure

```
switch (expression) {
    case value1 : 
        statement(s)
        break ;
    case value2 : 
        statement(s)
        break ;
    . . .
    default: 
        statement(s)
        break ;
}
```

Is this structure more efficient than the equivalent nested if-else structure?

switch Example

```
switch (day) {
    case 0: alert("Sunday");
    break ;
    case 1: alert("Monday");
    break ;
    case 2: alert("Tuesday");
    break ;
    case 3: alert("Wednesday");
    break ;
    case 4: alert("Thursday");
    break ;
    case 5: alert("Friday");
    break ;
    case 6: alert("Saturday");
    break ;
    default: alert("Error -- invalid day.");
    break ;
}
```

switch Statement Details

- The last statement of each case in the switch should *almost* always be a break.
- The break causes program control to jump to the closing brace of the switch structure.
- Without the break, the code flows into the next case. This is almost never what you want.
- A switch statement will work without a default case, but always consider using one.
Good Programming Practices

- Include a default case to catch invalid data.
- Inform the user of the type of error that has occurred (e.g., "Error - invalid day.").
- If appropriate, display the invalid value.
- If appropriate, terminate program execution (discussed in CMSC 201).

Why Use a switch Statement?

- A switch statement can be more efficient than an if-else.
- A switch statement may also be easier to read.
- Also, it is easier to add new cases to a switch statement than to a nested if-else structure.

Logical Operators

- So far we have seen only simple conditions.  
  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.
  
  && is AND  if (x > 5 && y < 6)
  || is OR  if (z == 0 || x > 10)
  ! is NOT if (!(bob > 42))

Example Use of &&

```javascript
if(age < 1 && gender == "f")
{
    alert("You have a baby girl!");
}
```

Example Use of ||

```javascript
if(grade == "D" || grade == "F")
{
    alert("See you next semester!");
}
```

Truth Table for &&

<table>
<thead>
<tr>
<th>Expression₁</th>
<th>Expression₂</th>
<th>Expression₁ &amp;&amp; Expression₂</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.
Truth Table for $\lor$

<table>
<thead>
<tr>
<th>Expression(_1)</th>
<th>Expression(_2)</th>
<th>Expression(_1) $\lor$ Expression(_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>nonzero</td>
<td>1</td>
</tr>
<tr>
<td>nonzero</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>nonzero</td>
<td>nonzero</td>
<td>1</td>
</tr>
</tbody>
</table>

Exp\(_1\) $\&\&$ Exp\(_2\) $\&\&$ ... $\&\&$ Exp\(_n\) will evaluate to 1 (true) if only ONE subcondition is true.

Example Use of $\neg$

```
if(!(age >= 18)) /* same as (age < 18) */
{
    alert("Sorry, you can’t vote.");
}
else
{
    alert("You can vote.");
}
```

Truth Table for $\neg$

<table>
<thead>
<tr>
<th>Expression</th>
<th>$\neg$ 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>* / %</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>$\lor$</td>
<td>left to right</td>
</tr>
<tr>
<td>=</td>
<td>right to left</td>
</tr>
</tbody>
</table>

Some Practice Expressions

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

<table>
<thead>
<tr>
<th>Expression</th>
<th>True/False</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
</tr>
<tr>
<td>a + b</td>
<td></td>
</tr>
<tr>
<td>a $&amp;&amp;$ b</td>
<td></td>
</tr>
<tr>
<td>a $|$ b</td>
<td></td>
</tr>
<tr>
<td>i3c</td>
<td></td>
</tr>
<tr>
<td>i1c</td>
<td></td>
</tr>
<tr>
<td>a $&amp;&amp;$ lb</td>
<td></td>
</tr>
<tr>
<td>a &lt; b $&amp;&amp;$ b &lt; c</td>
<td></td>
</tr>
<tr>
<td>a &gt; b $&amp;&amp;$ b &lt; c</td>
<td></td>
</tr>
<tr>
<td>a =&gt; b $|$ b &gt; c</td>
<td></td>
</tr>
</tbody>
</table>

More Practice

- Given
  ```
  var a = 3, b = 7, c = 21;
  ```
  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)