Relational & Logical Operators, if and switch Statements

Topics

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

Relational Operators

<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
==	is equal to
!=	is not equal to

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

Practice with Relational Expressions

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

Expression	true/false	Expression	true/false
a < c		a + b >= c	
b <= c		a + b == c	
c <= a		a != b	
a > b		a + b != c	
b >= c			

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

```
var  a = 1, b = 2, c = 3 ;
var  x = 3.33, y = 6.66 ;
```

Expression	Numeric Value	True/False
a + b		
b - 2 * a		
c - b - a		
c - a		
y - x		
y - 2 * x		

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

```
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

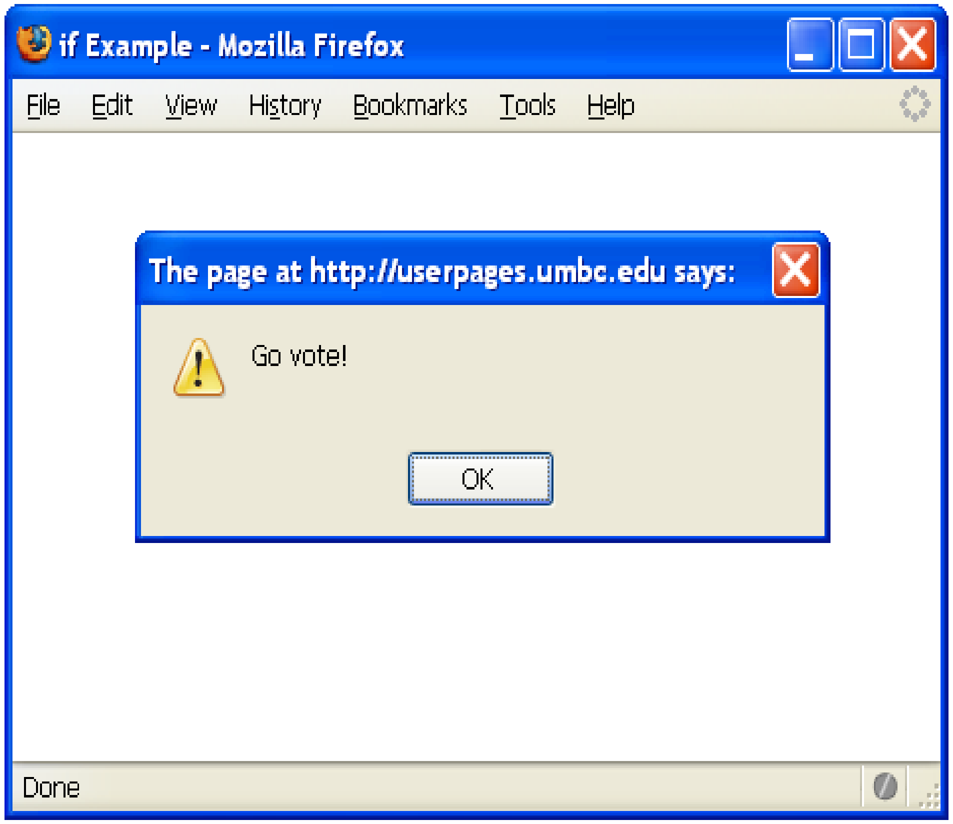
```
if(age >= 18)
{
   alert("Go Vote!");
}

if(value == 0)
{
   alert("You entered zero.");
}
```

Alert Screenshot

```
<script type="text/javascript">
  <!--
    var age = 18;

    if(age >= 18)
    {
       alert("Go Vote!");
    }
  //-->
</script>
```


if Example - Mozilla Firefox
File Edit View History Bookmarks Tools Help

The page at http://userpages.umbc.edu says:

Go vote!

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

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

- Note that there is no condition for the else.

Example

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

Another Example

```
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

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

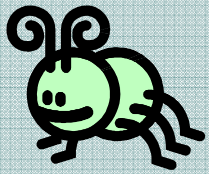
Another Example

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

Gotcha! = versus ==

```
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);
}
```



Multiple Selection with if

(continued)

```
if (day == 0 ) {
    alert ("Sunday") ;
}
if (day == 1 ) {
    alert ("Monday") ;
}
if (day == 2) {
    alert ("Tuesday") ;
}
if (day == 3) {
    alert ("Wednesday") ;
}
```

```
if (day == 4) {
    alert ("Thursday") ;
}
if (day == 5) {
    alert ("Friday") ;
}
if (day == 6) {
    alert ("Saturday") ;
}
if ((day < 0) || (day > 6)) {
    alert("Error - invalid day.") ;
}
```

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.") ;
}
```

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

The switch Multiple-Selection Structure

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

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 ;
}
```

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

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 &&

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

Truth Table for &&

$Expression_1$	$Expression_2$	$Expression_1$ && $Expression_2$
0	0	0
0	nonzero	0
nonzero	0	0
nonzero	nonzero	1

Exp_1 && Exp_2 && … && Exp_n will evaluate to 1 (true) only if ALL **subconditions** are true.

Example Use of ||

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

Truth Table for ||

Expression$_1$	Expression$_2$	Expression$_1$ || Expression$_2$
0	0	0
0	nonzero	1
nonzero	0	1
nonzero	nonzero	1

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

Example Use of !

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

Truth Table for !

Expression	! Expression
0	1
nonzero	0

Operator Precedence and Associativity

Precedence	Associativity
( )	left to right/inside-out
* / %	left to right
+ (addition) - (subtraction)	left to right
< <= > >=	left to right
== !=	left to right
&&	left to right
||	left to right
=	right to left

Some Practice Expressions

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

Expression	True/False
a	
b	
a + b	
a && b	
a || b	
!c	
!!c	
a && !b	
a < b && b < c	
a > b && b < c	
a >= b || b > c	

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)