CS 116 – Action Script Conditionals

Elie Abi Chahine

Conditionals

• Actionscript has conditional statements, also called selection statements.

• Essentially, depending on a certain condition, a program can decide which statements to execute and which ones to ignore.

• The simplest selection statement is the if statement:

```
if ( expression )
 {
      statement
 }
```

Note that the parentheses after the if keyword are required.

Conditionals

```
if ( expression )
{
      statement
}
```

You read this as:

"If expression is true, then execute statement.“

You could also read it as:

"If expression is false, then do not execute statement." (In which case statement is simply skipped.)

Expression

- *Expression is a boolean expression, meaning it is either **true** or **false**.*
- ***false** is also evaluated as a **zero** (0) and **non-zero** evaluates to **true**.*
- *To determine the value of the expression, you simply evaluate it.*
- *Assuming **a** is 5 and **b** is 0, these expressions are all true:*

a > b   a   a > 2   2 < a   b < 2   a - b   2   a * 5 * b + 4

*Assuming **a** is 5 and **b** is 0, these expressions are all false:*

b > a   b   a < 2   2 > a   a - 5   0   a * 5 * b

if statement

The if statement form is:

```
if(expression)
{
    statements
}
following statements
```

- If ‘expression’ is true then the statements are executed.

- If ‘expression’ is false then nothing is executed and the execution resumes at the following statement.

Eg:

```
var iNum:int = 5;
    if(iNum < 2)
    {
        trace(iNum);
    }
```

if statement

- An if statement can be nested.

Eg:
```
var iNum:int = 9;
if( iNum >= 9)
{
    iNum++;
    if( iNum < 10)   /* this is called a nested if*/
    {
        trace(iNum);
    }
}
```

if statement

- Example of bad code when not using braces

```
Eg:    var iNum:int = 10;
       if( iNum < 10)
              iNum++;                              /* Bad Code*/
              trace(iNum);
```

The output will be: 10 since the if condition only took **“*iNum++;*”** under it.

```
var iNum:int = 10;
if( iNum < 10)
{
    iNum++;                                        /* Good Code */
    trace(iNum);
}                                            /* Always use braces */
```

We won’t have an output since ***iNum*****<10** will return false

Operators and conditionals

- Relational operators:

<	**less than**
>	**greater than**
<=	**less than or equal to**
>=	**greater than or equal to**

- Equality operators:

==	**equal to**
!=	**not equal to**

Operators and conditionals

Ex:

```
var a:int = 5;
var b:int = 0;

trace("Value of a > b is " + (a > b) );
trace("Value of a == b is " + (a == b) );
trace("Value of a == a is " + (a == a) );
trace("Value of b == b is " + (b == b) );
trace("Value of a != a is " + (a != a) );
trace("Value of a > a is " + (a > a) );
```

Output:

```
Value of a > b is true
Value of a == b is false
Value of a == a is true
Value of b == b is true
Value of a != a is false
Value of a > a is false
```

Operators and conditionals

- Logical operators:

!	**Logical not (negation)**
&&	**Logical and**
||	**Logical or**

- Boolean truth table:

a	**b**	**a && b**	**a || b**
false	false	false	false
false	true	false	true
true	false	false	true
true	true	true	true

Operators and conditionals

<u>Notes about these operators:</u>

- Make sure you pay attention to the precedence of the operators.

- All the expressions will evaluate to 0 or 1 (false or true).

- The logical operators perform short circuit evaluation, meaning, as soon as the result can be determined, the evaluation stops.

- In English, this means:

  ❖ True or anything is true. (Short circuit: Won't bother evaluating anything)
  ❖ False and anything is false. (Short circuit: Won't bother evaluating anything)
  ❖ False or anything is anything. (Must evaluate anything)
  ❖ True and anything is anything. (Must evaluate anything)

<u>NB: If you don’t want to deal with all this, use parentheses ☺</u>

Operators and conditionals

Ex:

```
var a:int = 5;
var b:int = 3;

  if (a > b && b > 0 && ++a == 6)
    trace("1. The value of a is " + a);

  a = 5;
  if (a > b && b > 5 && ++a == 6)
    trace("2. The value of a is " + a);

  a = 5;
  if (a > b || b > 5 || ++a == 6)
    trace("3. The value of a is " + a);

  a = 5;
  if (a > b && b > 5 || ++a == 6)
    trace("4. The value of a is " + a);
```

Output:

```
1. The value of a is 6
3. The value of a is 5
4. The value of a is 6
```

if…else statement

The if else statement form is:

```
if(expression)
{
  statement1
}
else
{
  statement2
}
following statements
```

•If ‘expression’ is true

- Then statement1 is executed.
- statement2 is not executed. Execution resumes at “following statements”

•If ‘expression’ is false then statement2 is executed.

if…else statement

Eg:

```
if (x > 20)
{
    trace("x is > 20");
}
else
{
    trace("x is <= 20");
}
```

NB: The 'else' is associated with the closest previous non else if at the same block level.

```
if (x > 20)
if(x < 60)
else                    /* the else is for the closest if */
    trace("x is <= 20");
```

/* BAD CODE Use braces*/

if…else if statement

The if else statement form is:

```
if(expression1)
{
  statement1
}
else if (expression2)
{
  statement2
}
following statements
```

•If ‘expression1’ is true

- Then statement1 is executed.
- Execution resumes at “following statements”

•If ‘expression’ is false then:

- expression 2 is checked
- if expression2 is true then statement2 is executed.
- If expression2 is false, execution resumes at “following statements”

if…else if statement

Eg:

```
if (x > 20)
{
      trace("x is > 20");
}
else if (x < 0)
{
      trace("x is negative");
}
else
{
     trace (" x is between 0 and 20 );
}
```

For x = 30 output will be “x is > 20”
For x = -5 output will be “x is negative”
For x = 15 output will be “x is between 0 and 20”

The Conditional Operator

expression1 ? expression2 : expression3

This reads as:

If **expression1** is **true**, then execute **expression2**, otherwise (**else**), execute **expression3**

The following two examples are the same:

```
if (a > b)
{
      trace("a is larger\n");
}
else
{
      trace("a is NOT larger\n");
}
```

and

```
(a > b) ? trace("a is larger\n") : trace("a is NOT larger\n");
```

Switch Statement

- The switch statement form is:

```
switch (expression)
{
  case constant: statements
  case constant: statements
  :
  :
  default : statements
}
```

- The switch statement is designed to replace a cascade of if, else if, else ... statements.

Switch Statement

Eg:

```
if ( a==1 )
{
  statements
}
else if ( a ==2 )
{
  statements
}
else
{
  statements
}
```

```
switch (a)
{
    case 1:   /* checking if a==1 */
        statements
    break;

    case 2:  /* checking if a == 2 */
        statements
    break;

    default:   /* this will be my else cond */
        statements
    break;
}
```

Switch Statement

What happens if I don’t use a break at the end of the case?

```
switch( a )
{
    case 1:             /* since this case doesn’t have a break, it won’t stop
    trace(“1”);            at case 1 statements. It will run case 2 statements
                           as well */
    case 2:
    trace (“2”);
    break;

    case 3:
    trace (“3”);
    break;
}
```

If a is equal to 1 the output will be:
1
2

if a is equal to 2 the output will be: **2**

Switch Statement

Now How would I do the following: if (a==2 || a==3) ?

```
switch( a )
{
    case 1:
        trace("1");
        break;

    case 2:     /* this is equivalent to an || (or) */
    case 3:
        trace ("2");
        trace ("3");
        break;

    default:
        trace("default");
        break;
}
```

Conditionals

<u>Guides to writing good code:</u>

• The Number one rule is to use braces. I **DON'T** want to see any if, else, if-else or switch statement without open and closed braces.

• Conditional expressions that include negations are always hard to understand. Removing the negation makes the code read more naturally. Consider the following expression:

```
if (!(force < minForce) && !(force >= maxForce))
…
```

• Each test is stated with a negation, though there is no need for either to be. Changing the relations around lets us state the tests positively without a negation:

```
if ((force >= minForce) && (force < maxForce))
…
```

The End ☺