

Boolean Algebra

© A+ Computer Science - www.apluscompsci.com

George Boole

George Boole's work is considered by many the starting point of Boolean Algebra. His work is also considered as a beginning of sorts for Comp Sci.

Alice in Wonderland
Lewis Carroll

© A+ Computer Science - www.apluscompsci.com

George Boole is credited with laying the foundation for Boolean Algebra. The work of George Boole is still very important today.

What is a boolean?

A boolean is any condition or variable that can be evaluated to true or false.

```
boolean stop = false;  
boolean go = true;
```

```
if(x>10) { }
```


```
while(z<20) { }
```

© A+ Computer Science - www.apluscompsci.com

A boolean is anything that can be evaluated as true or false. Boolean variables can store the value true or false. Ifs and Loops have boolean conditions that are evaluated to true or false.

Operator Precedence

()	HIGH
! ++ --	
* / %	
+ -	
<< >> (bitwise shifts)	
< <= > >=	
== !=	
& (bitwise and)	
^ (bitwise xor)	
(bitwise or)	
&& (logical and)	
(logical or)	
= += -= *= /= %=	
,	LOW



Common Boolean Symbols

Name	Boolean Symbol	Java Counterpart
and	\wedge logical and	&&
or	\vee logical or	
not	\neg logical not	!

© A+ Computer Science - www.apluscompsci.com

And is used to see if all parts are true. In some languages, and is actually written as a word. In other languages, and is written as a symbol, like && or &.

Or is used to see if any part is true. In some languages, or is actually written as a word. In other languages, or is written as a symbol, like || or |.

Not is used to negate a boolean value. In some languages, not is actually written as a word. In other languages, not is written as a symbol, like !.

AND

&&

all conditions must be true

```
if (total==17 && 92==num)
{
    do something 1;
    do something 2;
}
```

© A+ Computer Science - www.apluscompsci.com

And is used to see if all parts are true. In some languages, and is actually written as a word. In other languages, and is written as a symbol, like && or &.

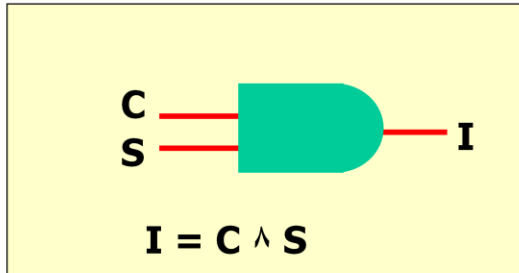
&& evaluates as true if all parts connected by &&s are true.

```
if (A and B)
```

This condition is true if A and B are both true. If either A or B is false, the condition is false as both parts must be true in order for the condition to be true.

AND

Engineering Symbol



C	S	I
0	0	0
0	1	0
1	0	0
1	1	1

OR

||

any condition can be true

```
if (total==9 || num==31)
{
    do something 1;
    do something 2;
}
```

© A+ Computer Science - www.apluscompsci.com

Or is used to see if any part is true. In some languages, or is actually written as a word. In other languages, or is written as a symbol, like || or |.

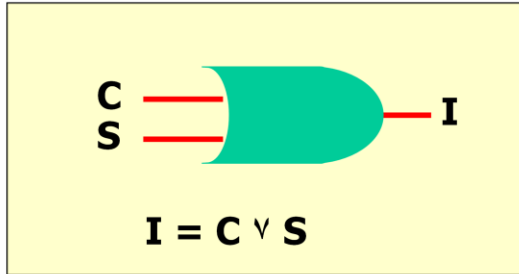
|| evaluates as true if any part connected by ||s is true.

```
if (A or B)
```

This condition is true if A or B is true. If A and B are both true, the condition is still true.

OR

Engineering Symbol



C	S	I
0	0	0
0	1	1
1	0	1
1	1	1

Fundamental Boolean Logic

true and false = false
false and true = false
false and false = false
true and true = true

false or true = true
true or false = true
true or true = true
false or false = false

© A+ Computer Science - www.apluscompsci.com

NOT

!

true (if condition is false)

```
if (! pass.equals("pass"))  
{  
    do something 1;  
    do something 2;  
}
```

© A+ Computer Science - www.apluscompsci.com

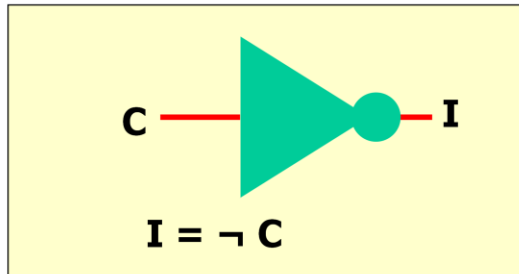
Not is used to negate a boolean value. In some languages, not is actually written as a word. In other languages, not is written as a symbol, like !.

!true is false

!false is true

NOT

Engineering Symbol



C	I
0	1
1	0

XOR

^

true if only one condition is true

```
if (total==34 ^ num==23)
{
    do something 1;
    do something 2;
}
```

© A+ Computer Science - www.apluscompsci.com

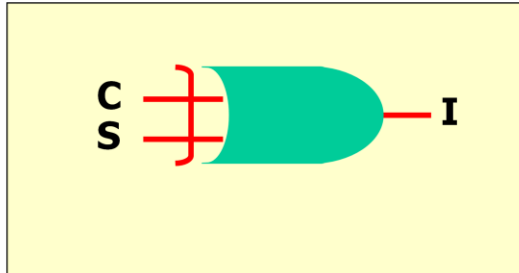
Xor is a very interesting boolean operator. Xor is true if any part of the condition is true, but false if more than one part is true.

```
if (A xor B)
```

This condition is true if A or B is true. If A and B are both true, the condition is false. If A and B are both false, the condition is false.

XOR

Engineering Symbol



C	S	I
0	0	0
0	1	1
1	0	1
1	1	0

Open logical.java

© A+ Computer Science - www.apluscompsci.com

Open
dowhile.java
password.java

© A+ Computer Science - www.apluscompsci.com

Common Boolean Laws

© A+ Computer Science - www.apluscompsci.com

Absorption Law

$$C \wedge (C \vee S) = C$$

Law of Absorption

$$C \vee (C \wedge S) = C$$

Law of Absorption

`c&&(c || s)`

Java Code

`c || (c&&s)`

Java Code

This is used now and again by AP and UIL!

© A+ Computer Science - www.apluscompsci.com

The law of absorption is essentially simplifying the expression. C is the dominant term in the expressions above. C alone determines if the expression is true or false.

Boolean Example 1

```
boolean c = true;  
boolean s = false;  
boolean i = c || (c&&s);  
System.out.println(i);
```

c	s	i
1	1	1
1	0	1
0	1	0
0	0	0

OUTPUT

true

Boolean Example 2

```
boolean c = false;  
boolean s = true;  
boolean i = c && (c || s);  
System.out.println(i);
```

OUTPUT

false

c	s	i
1	1	1
1	0	1
0	1	0
0	0	0

open
absorptionlaw.java

© A+ Computer Science - www.apluscompsci.com

Distributive Law

$$\begin{aligned} C \wedge (S \vee I) &= (C \wedge S) \vee (C \wedge I) && \text{Distributive} \\ C \vee (S \wedge I) &= (C \vee S) \wedge (C \vee I) && \text{Distributive} \end{aligned}$$

$c \& (s \mid i)$
is the same as
 $(c \& s) \mid (c \& i)$

This is used now and again by AP and UIL!

© A+ Computer Science - www.apluscompsci.com

The Distributive Law is a basic algebraic law.

In the expressions above, distributing the term on the outside of parenthesis to each term inside the parenthesis is equivalent to the original expression.

Boolean Example 3

```
boolean c=true,s=true,i=false,ans;  
ans=((c || (s&& i)) == ((c || s) && (c || i)));  
System.out.println(ans);
```

OUTPUT

true

open
distributivelaw.java

© A+ Computer Science - www.apluscompsci.com

Start work on Lab 10

© A+ Computer Science - www.apluscompsci.com

More Boolean Laws

© A+ Computer Science - www.apluscompsci.com

DeMorgan's Law

$$\neg(C \vee S) = \neg C \wedge \neg S$$

DeMorgan's Law

$$\neg(C \wedge S) = \neg C \vee \neg S$$

DeMorgan's Law

`!(c | s) == !c&&!s`

Java Code

`!(c&& s) == !c | !s`

Java Code

This is always used by AP and UIL!

© A+ Computer Science - www.apluscompsci.com

Demorgan's Law is useful to get a better picture of how the not affects the condition.

The easiest way to evaluate a negated condition is to simply evaluate the condition and then apply the negation.

If the condition is true, applying the negation makes the condition false.

If the condition is false, applying the negation makes the condition true.

AND

&&

all conditions must be true

```
if (c==true && s==true)
{
    do something 1;
    do something 2;
}
```

C	S	I
0	0	0
0	1	0
1	0	0
1	1	1

Boolean Example 4

```
boolean c = true;  
boolean s = true;  
boolean i = !(c&&s);  
System.out.println(i);
```

c	s	i
1	1	0
1	0	1
0	1	1
0	0	1

OUTPUT

false

Boolean Example 5

```
boolean c = false;  
boolean s = true;  
boolean i = !(c | s);  
System.out.println(i);
```

c	s	i
1	1	0
1	0	0
0	1	0
0	0	1

OUTPUT

false

open
demorganslaw.java

© A+ Computer Science - www.apluscompsci.com

Boolean Example 6

Which statement is represented by the truth table at right?

- A. $i = !(c \& \& s) \& \& (c | | s);$
- B. $i = c | | s \& \& s;$
- C. $i = c \& \& s;$
- D. $i \neq c \& \& s;$

c	s	i
0	0	0
0	1	1
1	0	1
1	1	1



Boolean Example 7

Which statement is represented by the truth table at right?

- A. $i = !(c \& \& s) \& \& c | | s;$
- B. $i = c | | s \& \& s;$
- C. $i = c \& \& s;$
- D. $i = !(c \& \& s) \& \& (c | | s);$

c	s	i
0	0	0
0	1	1
1	0	1
1	1	0

d

Short Circuit Evaluation

© A+ Computer Science - www.apluscompsci.com

Short Circuit Evaluation

Java evaluates boolean expressions from left to right in most situations and stops the evaluation process once a condition is found that can complete the expression.

&& - and

|| - or

© A+ Computer Science - www.apluscompsci.com

Short Circuit Evaluation || or

```
int total=9;  
boolean flipper = false;
```

```
if(flipper || total>4)  
{  
    out.println("short");  
}  
out.println("check");
```

OUTPUT

**short
check**

© A+ Computer Science - www.apluscompsci.com

If flipper is true, total>4 is never evaluated.

If flipper is false, total>4 is evaluated.

flipper is false, but total>4 so the condition is true. short is printed.

Short Circuit Evaluation || or

```
int total=2;  
boolean flipper = true;
```

```
if(flipper || total>4)  
{  
    out.println("short");  
}  
out.println("check");
```

OUTPUT

**short
check**

© A+ Computer Science - www.apluscompsci.com

If flipper is true, total>4 is never evaluated.

If flipper is false, total>4 is evaluated.

flipper is true so the condition is true. short is printed.

Short Circuit Evaluation || or

```
int total=2;  
boolean flipper = false;
```

```
if(flipper || total>4)  
{  
    out.println("short");  
}  
out.println("check");
```

OUTPUT

check

© A+ Computer Science - www.apluscompsci.com

If flipper is true, total>4 is never evaluated.

If flipper is false, total>4 is evaluated.

flipper is false and total>4 is false so the condition is false.
short is not printed.

Short Circuit Evaluation || or

```
int total=9, num=13;  
  
if (total<4 || ++num<15)  
{  
    out.println("short");  
}  
out.println(num);
```

OUTPUT

short
14

© A+ Computer Science - www.apluscompsci.com

If `total<4` is true, `++num<15` is never evaluated.

If `total<4` is false, `++num<15` is evaluated.

`total<4` is false so `++num<15` is evaluated. 14 is less than 15 so short is printed.

Short Circuit Evaluation && and

```
int total=9, num=13;  
  
if (total>4 && ++num>15)  
{  
    out.println("short");  
}  
out.println(num);
```

OUTPUT

14

© A+ Computer Science - www.apluscompsci.com

If `total>4` is false, `++num<15` is never evaluated.

If `total>4` is true, `++num<15` is evaluated.

`total>4` is true so `++num<15` is evaluated. 14 is less than 15 so short is not printed.

Short Circuit Evaluation && and || or

```
int total=9, num=13;
```

```
if (total>4 || ++num>15 && total>0)
{
    out.println("short");
}
out.println(num);
```

OUTPUT

**short
13**

The && never happens!

open
shortone.java
shorttwo.java
shortthree.java
shortfour.java

No-Short Circuit Evaluation

© A+ Computer Science - www.apluscompsci.com

No Short Circuit Evaluation

Java also has logical operators that do not use short circuit evaluation.

& - and

| - or

No Short Circuit Evaluation & and

```
int x=9;  
int y=10;
```

```
if(x == 10 & ++y>=11)  
    out.println("no short 1");  
out.println("y == " + y);
```

OUTPUT

y == 11

© A+ Computer Science - www.apluscompsci.com

If `x==10` and `y>=11`, the condition is true.

If `x==10` is false, `++y` still happens as `&` is a non-short circuit operator.

No Short Circuit Evaluation | or

```
int x = 9;  
int y = 10;
```

```
if(y>5 | x==10 & ++y>=11 )  
    out.println("no short 2");  
out.println("y == " + y);
```

OUTPUT

```
no short 2  
11
```

No Short Circuit Evaluation | or

```
int x = 9;  
int y = 10;
```

```
if(y>10 | x==10 & ++y>=11 )  
    out.println("no short 3");  
out.println("y == " + y);
```

OUTPUT

y == 11

open
noshortone.java
noshorttwo.java
noshortthree.java

Random Numbers

© A+ Computer Science - www.apluscompsci.com

Math.random()

```
double decOne;  
decOne = Math.random() * 10;  
int intOne;  
intOne = (int)(Math.random() * 10);  
  
System.out.println(decOne);  
System.out.println(intOne);
```

OUTPUT

```
8.44193167660682  
6
```

© A+ Computer Science - www.apluscompsci.com

Math.random() returns a number between 0.0 and 1.0, not including 1.0.

Random Instantiation

reference variable

```
Random rand = new Random();
```

object instantiation

Always make Random vars instance vars!

© A+ Computer Science - www.apluscompsci.com

Class Random contains many useful random methods. Math.random() can be used to generate the same results as the Random class methods.

Random frequently used methods

Name	Use
nextInt(x)	returns a random int 0 to x(exclusive)
nextInt()	returns a random int MIN to MAX(exclusive)
nextDouble()	returns a random int 0.0 to 1.0(exclusive)

```
import java.util.Random;
```

Random

```
Random rand = new Random();  
int intOne = rand.nextInt(10); //0-9  
System.out.println(intOne);  
intOne = rand.nextInt(50)+1; //1-50  
System.out.println(intOne);  
intOne = rand.nextInt(20)+20; //20-39  
System.out.println(intOne);
```

OUTPUT

```
7  
29  
37
```

open
randomone.java

© A+ Computer Science - www.apluscompsci.com

Continue work on Lab 10

© A+ Computer Science - www.apluscompsci.com