BOOLEAN HAIR LOGIC

A

B

AND

OR

XOR
; and if

Please always put {} after if-statements

The compiler will let you get away with not putting these (this leads to another issue)

If you do not put {} immediately after an if, it will only associate the first command after with the if-statement (see: ifAndSemi.cpp)
Logical operators

These are all the operators that result in a `bool`:

> (greater than), e.g. 7 > 2.5 is `true`
== (equals), e.g. 5 == 4 is `false`
< (less than), e.g. 1 < 1 is `false`
>= (greater than or equal to), e.g. 1 <= 1 is `true`
!= (not equal to), e.g. 8 != 7 is `true`
<= (less than or equal to), e.g. 6 <= 2 is `false`
! (not, negation), e.g. !true is `false`
Random numbers

To use random numbers, you need to do:
1. Run srand(time(0)) once
2. Use rand() to actually generate a number

```cpp
int main()
{
    srand(time(0));
    cout << rand()%10 << endl; // displays 0-9
}
```

(See: rng.cpp)
Complex expressions

Two boolean operators:
&& is the AND operations
|| is the OR operations
Complex expressions

AND operation removes Ts from the result
The OR operation adds Ts to the result

Evaluate (!p OR q) AND (p)

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>!p</th>
<th>!p OR q</th>
<th>(!p OR q) AND (p)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>
Write an if statement for checking if a variable (int) x is a positive odd number.

Hint: You may want to use the remainder (also called modulus) operator (the % sign).

For example, 5 % 3 = 2
int x = 9, y = 7;

Complex expressions

\[
\begin{align*}
x & < 12 & \& \& y & > 10 \\
9 & < 12 & \& \& 7 & > 10 \\
T & \& \& F \\
& \\& \& F
\end{align*}
\]
Complex expressions

Write boolean expressions for each of the following truth tables:

1. 
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

2. 
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

3. 
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

4. 
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

XOR
Complex expressions

Humans tend to use the English word OR to describe XOR (exclusive or)

“You can get a side order of a salad, fries or a soup.”

Did you think the statement above meant you could get all three?
Complex expressions

If statements for when $x$...

... is between 10 and 20 (inclusive)

```java
if(10 <= x && x <= 20)
```

Cannot say: $10 \leq x \leq 20$ (why?)

... is a vowel ($x$ is type `char`)

```java
if( x == 'a' || x == 'e' || x == 'i' || x == 'o' || x == 'u')
```
Short-circuit evaluation is when you have a complex bool expression (&& or ||) but you don't need to compute all parts.

```cpp
if(false && 7/0 == 2) {
    cout << "Will I crash?\n";
}
```

If this is false, then it will not check next.

(See: shortCircuit.cpp)
Short-circuit evaluation

Simple cases of short-circuit:
When you have a bunch of ORs
if( expression || exp || exp || exp )
Once it finds any true expression, if statement will be true

When you have a bunch of ANDs
if( expression && exp && exp && exp )
Once it finds any false expression, if statement will be false
Complex expressions

Be careful when negating, that you follow De Morgan's Law:

```cpp
bool a, b;
!(a OR b) is equivalent to (!a) AND (!b)
!(a AND b) is equivalent to (!a) OR (!b)
```

“Neither rainy or sunny” means
“Both not rain and not sunny”
Nested if statements

You can have as many if statements inside each other as you want.

```java
if (teacherAwake)
{
    if (studentAwake)
    {
        if (classWellPrepared)
        {
            learning = true;
        }
    }
}
```
Nested if statements

From a truth table perspective, nested loops are similar to AND

The previous if code is equivalent to:

```java
if(teacherAwake && studentAwake && classWellPrepared)
{
    learning = true;
}
```

However, sometimes you want to do other code between these evaluations.
Nested if statements

(See: bridgeOfDeath.cpp)
Scope

Where a variable is visible is called its scope.

Typically variables only live inside the block (denoted with matching { and }).

A variable lives until the block is closed, so inner blocks can see everything from the block it was created inside.
int main()
{
    int x;
    // can use x here
    {
        int y;
        // can use x or y here
    }
    // can use x here
    return 0;
}

(See: scope.cpp)
If... if... else!

if there were two ifs and one else
To which if does else belong to
If... if... else!

When in doubt, use parenthesis and blocks! (Some people like to put the first brace after the if, others on a new line)

What happens if you have an if if else?

(See: ifIfElse.cpp)
Multiway if/else

This is a special format if you put an if statement after an else.

This second “if statement” only is tested when the first “if statement” is not true.

(See: grades.cpp)
A **switch statement** checks to see if a variable has a specific value.

```cpp
switch( controllingVariable)
{
    case 2:
    case 4:  // Case label
        cout << "controllingVariable is either 2 or 4" << endl;
        break;
    case 3:  // Case label
        cout << "controllingVariable is 3\n";
        break;
    default;
        cout << "controllingVariable is not 2, 3 or 4...\n";
        break;
}
```
Switch

If the value of the controlling variable is found in a case label, all code until a break statement is ran (or the switch ends).

Switch statements only test equality with case labels (not greater or less than).

(See: switch.cpp)
Switch

Switch statements can be written as multiway if/else statements.

Could use just “if statements” but “else if” shows only one of these will run

(See: switchToIf.cpp)
Conditional operator

We will not use in this class, but if you use other people's code you will encounter

Shorthand for an if-else statement

(boolean) ? [if true] : [if false]

Example:
max = (x>y) ? x : y;
(See: max.cpp)