Branching

I'll be in your city tomorrow if you want to hang out.

But where will you be if I don't want to hang out?!

You know, I just remembered I'm busy.

Why I try not to be pedantic about conditionals.
Announcements

Homework 0: using cin with 10/3 is NOT the same as 3.3333 (directly)

With cin, it will stop as soon as it reaches a type that does not match the variable (into which it is storing)

(See: cinMismatchTypes.cpp)
Outline

if/else statements cont.
boolean expressions
bool

- either true or false

C++ lets you change between fundamental types (casting) with ease

Q: 0 is false and 1 is true, right?
A: 1.
boolean values

Sometimes this might cause an error, such as:

```java
int x = 2;
if( ! x>5 ) will be false
```

Why?
boolean values

Sometimes this might cause an error, such as:

```java
int x = 2;
if( ! x>5 ) will be false
```

Why?
A: order of operations will do the unary operator first (the '!'')
if (! x>5) will become if ( (!2) > 5) 
... if ( (!true) > 5) ... if ( false > 5) ... if (0 > 5)
if statement

Code inside an if statement is only run if the condition is true.

Indent

(See last week: numberGuessing.cpp)
if/else statement

Immediately after an if statement, you can make an else statement.

If the “if statement” does not run, then the else statement will.

If you do not surround your code with braces, only one line will be in the if (and/or else) statement.
if/else statement

(See: ATM.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
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></th>
<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></td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td></td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td></td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td></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, \( 5 \% 3 = 2 \)
Humans tend to use the english word OR to describe XOR (exclusive or)

“We can have our final exam on the scheduled day (May 13) or the last day of class (May 6).”

Did you think the statement above meant final exams on both days was a possibility?
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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

2. XOR

<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>

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>0</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>
Complex expressions

```c
int x = 9, y = 7;
```

\[
x < 12 \land \land y > 10
\]

\[
9 < 12 \land \land 7 > 10
\]

\[
T \land \land F
\]

\[
F
\]
Complex expressions

If statements for when x...

... is between 10 and 20 (inclusive)

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

Cannot say: 10 <= x <= 20 (why?)

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

```
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

Write a single if-statement that is true on the following range of numbers:
sample) **int** i: 3
   Answer: if( i == 3)
a) **int** i: ... -2, -1, 0
b) **int** i: 5, 6, 7, 8, ...
c) **int** i: 1, 2, 3, 4, 5
d) **int** i: ... -2, -1, 1, 2, 3, ...
e) **int** i: ... -2, -1, 5, 6, 7, ...
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”