Boolean operators

... and other miscellaneous logic stuff
Goals

By the end of this lesson you will be able to:

- Understand Boolean operators and precedence
- Understand short-circuit Boolean operator evaluation
- Understand and use Boolean return values of some objects and functions
- Understand the ternary (conditional) operator `?:`
- Understand `switch/case` structures
- Understand the `assert` function
Boolean operators

- Boolean operators can be used to combine logical statements
  - ! Boolean (logical) **not**, a unary operator
  - && Boolean (logical) **and**, a binary operator
  - || Boolean (logical) **or**, a binary operator

- The next slide contains **truth tables** for these operators

- Similar to how = and == are different, do not mix up && and || with & and | which are different things called bitwise operators
# Truth tables

<table>
<thead>
<tr>
<th>A</th>
<th>!A</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A &amp;&amp; B</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

| A   | B   | A || B |
|-----|-----|------|
| false| false| false|
| false| true | true |
| true | false| true |
| true | true | true |
Operator precedence

- Highest precedence is !
- Next highest precedence is &&
- Lowest precedence is ||
- It is highly suggested to use parentheses to increase readability and reduce ambiguity
- Note that the precedence of these operators is also mixed in with all the other operators you know
Short-circuit evaluation

• If a logical expression is guaranteed to evaluate to true or false, the rest will not be evaluated

• For example if ( (5 > 3) || (x < 20) ) will not ever evaluate the x < 20 part – this could be relevant if you are doing more complicated things such as increment
Stream logic

- Logic related to streams can be useful, for example:
- An `istream` variable (e.g. `cin` or an `ifstream` variable) evaluates to true if the stream is in a good state and there exists data to be read
- `istreamvar.good()` evaluates to true if the stream is in a good state
- `ifstreamvar.is_open()` evaluates to true if a file was opened successfully
The ternary, or conditional, operator `?:` is the only C++ operator that takes three arguments and it acts similar to an if statement.

`a ? b : c` is equivalent to

```cpp
if (a)
    b;
else
    c;
```

It should be used very sparingly.
Often when you would have many options from which to choose, a `switch/case` structure is a better option.

This is best used when you have a non-trivial number of potential options and associated code for each, such as a menu system.

The next slide contains the relevant syntax and the equivalent way of doing it with `if` statements.
switch (ex) {
    case value1:
        statements_A;
        break;
    case value2:
        statements_B;
        break;
    default:
        statements_C;
}
switch/case

- The `break` statement at the end of each `case` is needed, otherwise the code in the next `case` will continue running (this may be desirable, but usually isn’t)
- You can have as many `case` statements as needed, and the `default` statement is optional
assert

- The `assert` function, available from the header file `cassert`, can be used to terminate your program if the assertion fails.

- `assert(expr)` evaluates the expression logically and, if the expression evaluates to false, exits your program immediately.

- There are many ways of “error-proofing” your code, and this is one of them although it is perhaps the least graceful.
Exploration

• Write your own program that utilizes the Boolean operators, perhaps something involving a user who enters their age and the amount of money they have, and then suggests options for things they can do.

• Write a menu system similar to the last video, but this time do it with a `switch/case` structure.