Logic and Conditions

CS 1044

Boolean Logic

- Boolean algebra has just two values: `true` and `false`
- Represented by `bool` type in C++

```
bool x = true;    bool y = false;
```

- Used in programming to answer **yes/no questions**:
  - Is something true? (Yes it is, or no it isn't)
  - Did something happen? (Yes it did, or no it didn't)
  - ... and so on

Relational Operators

- C++ has six **relational operators** that are used to compare values and produce a `true` or `false` result

==	Equals	!=	Does not equal
<	Less than	<=	Less than or equal
>	Greater than	>=	Greater than or equal

```
int x = 5;
bool a = (x == 6);
bool b = (x > 3);
bool c = (x != 9);
```

Logical Operators

- C++ also has three **logical operators** that are used to combine boolean expressions

`&&`	And	`(p && q)` is true if both `p` and `q` are true
`||`	Or	`(p || q)` is true if `p` is true, or `q` is true, or **both** are true
`!`	Not	`!p` is the opposite of `p`

Truth Tables

a	b	a && b
F	F	F
F	T	F
T	F	F
T	T	T

a	b	a || b
F	F	F
F	T	T
T	F	T
T	T	T

a	!a
F	T
T	F

Ambiguity of "Or" in English

- **Inclusive-or** (try inserting the words **"or both"**)
  - *"We'll give you $10 if you get an A in math or history."*
- **Exclusive-or** (try inserting the word **"either"**)
  - *"I want Italian or Mexican for dinner."*
- || always means **inclusive-or**, so be careful when translating your thoughts from English to C++

Tricky Details

- Remember the difference between = and ==
  - `x = y` "Set variable `x` to the value in variable `y`."
  - `x == y` "Is `x` equal to `y`?"
- You **cannot** combine multiple relational operators like you might see in mathematics
  - $0 \leq x < 10$ in C++: `0 <= x && x < 10`

Tricky Details

- Be careful using **multiple** ORs or ANDs together
- English: "*x equals 10, 20, or 30*"
- You might try to write this in C++ as

```
x == 10 || 20 || 30                    // wrong
```

- You must write each of the comparisons **individually**

```
x == 10 || x == 20 || x == 30      // good
```

Precedence

- `!` has higher precedence than `&&` and `||`

  `!p && q` means: *(NOT p) AND q*

  `!(p && q)` means: *NOT (p AND q)*

- `&&` has higher precedence than `||`

  `p || q && r` means: *p OR (q AND r)*

- It never hurts to **use parentheses** to clarify your intentions

Short Circuiting

- `&&` and `||` only evaluate their right-hand side if necessary – this is called **short circuiting**

```
false && p    true || p
```

`p` will never be evaluated

- We can conveniently write combined expressions that would fail if the whole thing was evaluated at once

```
n != 0 && x / n > 5
```

`x / n` will not be evaluated if `n == 0`

Control Flow

- **Sequencing**
  - “Do this, then do that”
- **Selection**
  - “If some condition is true, do this; otherwise, do that”
- **Iteration**
  - “While some condition is true, do this”
  - “Do this *X* number of times”

Conditions

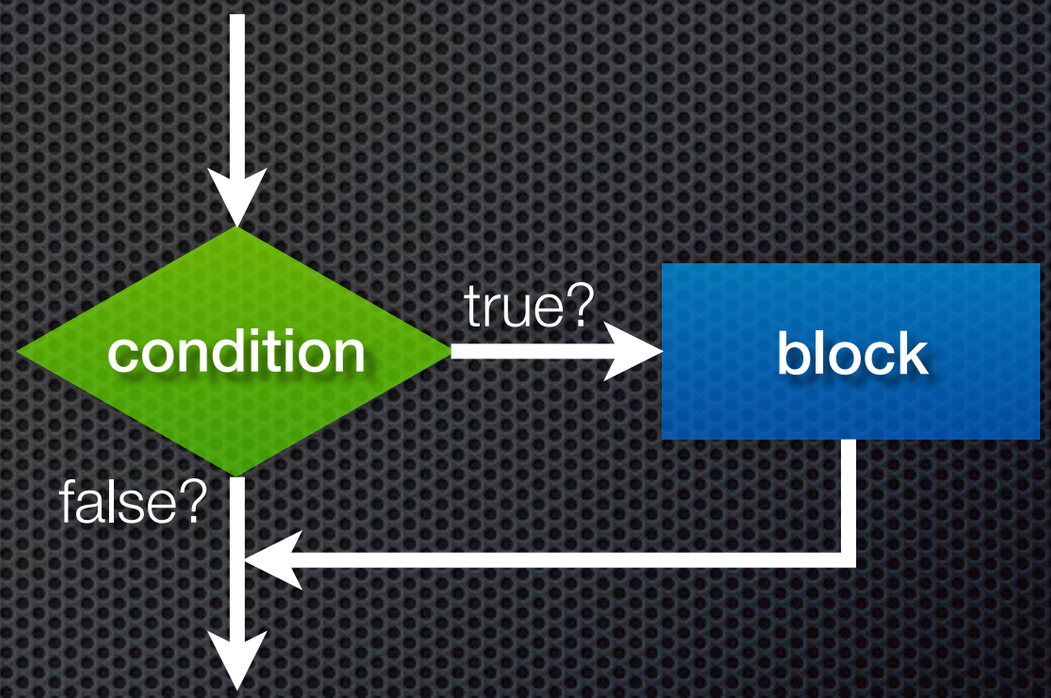
- Boolean expressions are most commonly used to **make decisions**, to **alter the flow** of the program

- We use Boolean expressions with **conditional statements** to execute one path if something is true, or another path if something is false

Simple `if` Statement

```
if (condition)
{
    block;
}
```

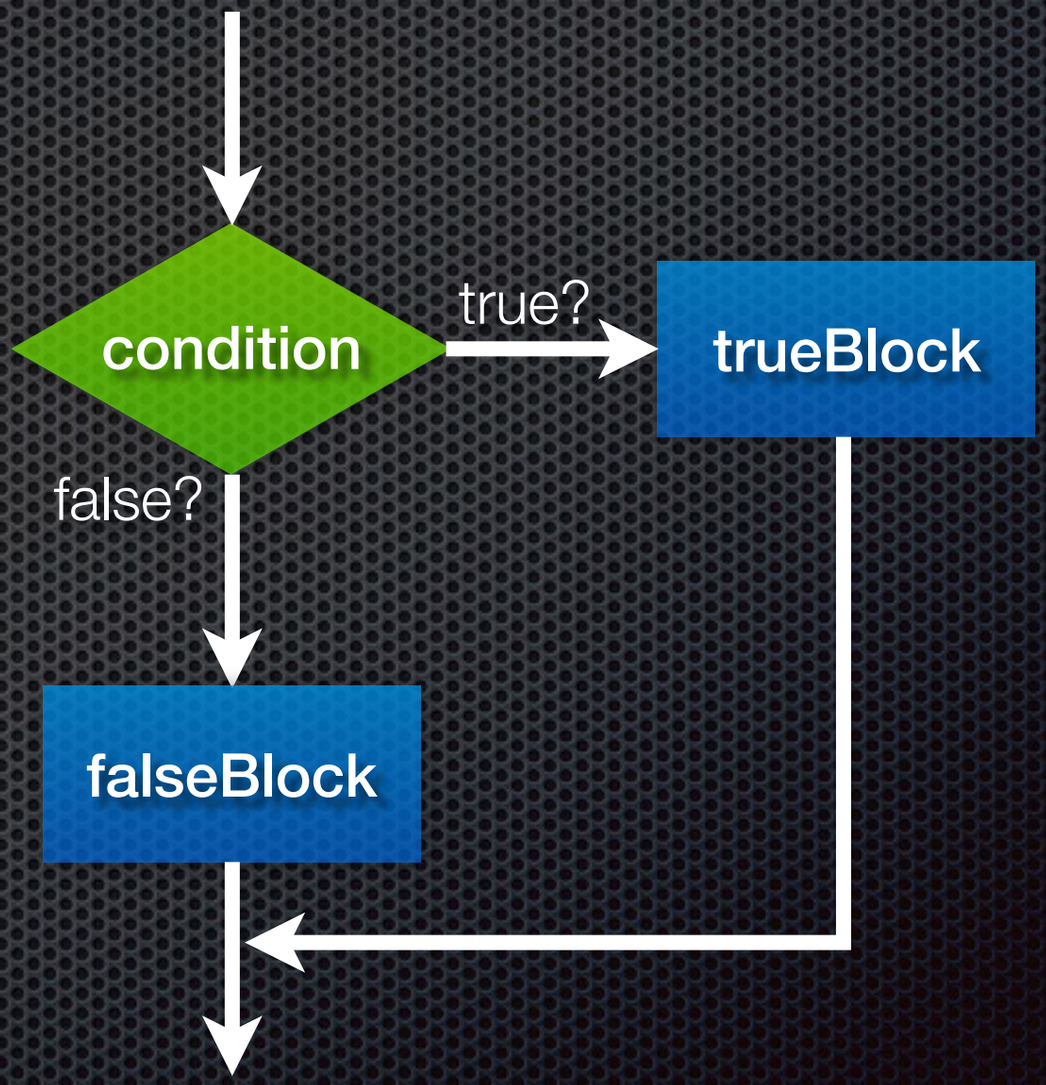
Notice there is **no semicolon** at the end of this line



`block` is **one or more statements** that will be executed, in this case, when *`condition`* is true

`if`/`else` Statement

```
if (condition)
{
    trueBlock;
}
else
{
    falseBlock;
}
```



Simple Example

```
int first = 15;
int second = 20;

if (first > second)
{
    cout << “The first number is bigger.”;
}
else
{
    cout << “The second number is bigger.”;
}
```

Multi-way `if` Statement

```
if (condition1)
{
    block1;
}
else if (condition2)
{
    block2;
}
... more else ifs ...
else
{
    elseBlock;
}
```

Must start with this

Can have as many of these as you like

Optional, gets executed if none of the others are true

Mutually exclusive: only one block will be executed – if multiple conditions are true, only the first one runs

`switch` Statement

```
switch (expression)
{
   case value:
      block;
      break;
   ...more cases...
   default:
      defaultBlock;
      break;
}
```

Can have as many of these as you like

Optional, gets executed if none of the others match

`expression` must be `int` or `char` (as far as we're concerned)
Case values **must be literals** (not variables/expressions)

Combining `case` Blocks

- Combine multiple cases to execute the same block for any of them
- In this example, *`block`* runs if expression equals *`value1`*, *`value2`*, or *`value3`*

```
switch (expression)
{
   case value1:
   case value2:
   case value3:
     block;
     break;
}
```

Nesting

- `if` and `switch` blocks can contain any kind of statements, so we can even **nest** them if we need to

```
if (x)
{
  if (y)
  {
    cout << “hello”;
  }
  else
  {
    cout << “goodbye”;
  }
}
```

What would be printed if...

- x is `true` and y is `true`?
- x is `true` and y is `false`?
- x is `false` and y is `true`?
- x is `false` and y is `false`?

`if` with Single Statements

- C++ lets you **omit** the curly braces after an `if`/`else` statement if what follows it is a **single** statement

```
if (x == 5)
    cout << “x was 5”;
```

- This is generally a **bad idea**
- Spacing and indentation **don’t matter** in C++, which makes it easy to **make mistakes** if you omit braces

`if` with Single Statements

- What does the following example print?

```
int x = 5;
if (x == 5)
    cout << "hello";
    cout << "goodbye";
```

```
cout << “Enter a number: “ << endl;
cin >> x;

// Only the first “cout” is part of the “if”.

if (x == 5)
    cout << "hello";
    cout << "goodbye";


// This might print “hello”
// but will always print “goodbye”.
```

```
cout << “Enter a number: “ << endl;
cin >> x;

// Add braces to include both lines in the “if”.
if (x == 5)
{
    cout << "hello";
    cout << "goodbye";
}

// You should always use braces.
```

```
cout << “Enter a number: “ << endl;
cin >> x;

// if / else if / else statements let you
// select between multiple choices.
if (x == 5)
{
    cout << "hello";

}
else if(x == 4)
{
    cout << “hi”;
}
else
{
    // This happens when the if and else if
    // conditions aren’t satisfied.
    cout << "goodbye";
}
```