The website for these SI sessions is `https://github.com/benblazak/2014-fall-si-cpsc120`.

Many of these examples are from `https://github.com/benblazak/2014-spring-si-cpsc120`, which is full of stuff I wrote for a lab last semester. If you're looking for extra practice, this is one of many places you might start.

Along with your book, `http://www.cplusplus.com/doc/tutorial/` is a great resource for tutorials, and `http://www.cplusplus.com/reference/clibrary/` is a great reference.

Syntax and Stuff

- What's the difference between the "declaration" and the "initialization" of a variable?
- What is the `cmath` library? Where can I learn about it?
- What is a binary operator? A unary operator?
- What is operator precedence? Associativity?
- What is the syntax for a `cout` statement (i.e. what are each of the pieces in a full statement beginning with `cout`)?
- What is the syntax for a `cin` statement?

Practice

- At the beginning of your program, you want a variable `a` equal to `5` and a variable `b` equal to `7`. What are three different ways to write this?

- Convert the following equations to C++ syntax:
  (a) $c = \sqrt{a^2 + b^2}$

  (a) $x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$

Program Design

From the Abacus International Math Challenge, for Grades 3–4:

> A.975. The Strongest Man of Etown had to roll a huge rock ball 10 meters. The spectators may guess the weight of the ball. Four people gave the following guesses to the organizers: 196 kg, 163 kg, 178 kg, and 185 kg. One guess was 1 kg off, one was 6 kg off, one was 17 kg off, and one was 16 kg off. How many kg was the weight of the ball?

How could we find the answer using C++?

Code

operators.cpp

```
#include <iostream>
using std::cout;
using std::endl;

int main() {
    cout << "5 / 2 % 3 * 7 / 3 = " << 5 / 2 % 3 * 7 / 3 << endl;
    cout << "2- -3-5           = " << 2- -3-5 << endl;
    cout << "4+ +7+2           = " << 4+ +7+2 << endl;
    cout << "4+-3-+2           = " << 4+-3-+2 << endl;

    return 0;  // success
}
```

```
5 / 2 % 3 * 7 / 3 = 4
2- -3-5           = 0
4+ +7+2           = 13
4+-3-+2           = -1
```

float.cpp

```
// Lots of good information about floating point numbers is on Wikipedia:
// http://en.wikipedia.org/wiki/Single-precision_floating-point_format

#include <iostream>
using std::cout;
using std::endl;

#include <iomanip>
using std::setprecision;

int main() {
    cout << "float(0)      = " << float(0) << endl;
    cout << "float(1)      = " << float(1) << endl;
    cout << "float(1e38)   = " << float(1e38) << endl;
    cout << "float(1e39)   = " << float(1e39) << endl;
    cout << "float(1e-39)  = " << float(1e-39) << endl;
    cout << "float(1e-40)  = " << float(1e-40) << endl;
    cout << "float(1e-45)  = " << float(1e-45) << endl;
    cout << "float(1e-46)  = " << float(1e-46) << endl;
    cout << endl;
    cout << setprecision(20) << "float(1234567890123467890) = "
         << float(1234567890123467890) << endl;

    return 0;  // success
}
```

```
float(0)      = 0
float(1)      = 1
float(1e38)   = 1e+38
float(1e39)   = inf
float(1e-39)  = 1e-39
float(1e-40)  = 9.99995e-41
float(1e-45)  = 1.4013e-45
float(1e-46)  = 0

float(1234567890123467890) = 1234567939550609408
```

typecast.cpp

```
#include <iostream>
using std::cout;
using std::endl;

int main() {
    cout << "'a'              = " << 'a' << endl;
    cout << "char(97)         = " << char(97) << endl;
    cout << "'a'+5            = " << 'a'+5 << endl;
    cout << "char(97)+5       = " << char(97)+5 << endl;
    cout << "char('a'+5)      = " << char('a'+5) << endl;
    cout << "char(char(97)+5) = " << char(char(97)+5) << endl;
    cout << "char(97)+char(5) = " << char(97)+char(5) << endl;
    cout << endl;
    cout << "int('a')    = " << int('a') << endl;
    cout << "double('a') = " << double('a') << endl;
    cout << "int(97)     = " << int(97) << endl;
    cout << "double(97)  = " << double(97) << endl;
    cout << "int(5.7)    = " << int(5.7) << endl;
    cout << "double(5.7) = " << double(5.7) << endl;
    cout << "char(98)    = " << char(98) << endl;
    cout << "char(98.6)  = " << char(98.6) << endl;
    cout << endl;
    cout << "1/2                    = " << 1/2 << endl;
    cout << "1/2.0                  = " << 1/2.0 << endl;
    cout << "2147483647 + 10        = " << 2147483647 + 10 << endl;
    cout << "2147483647 + 10L       = " << 2147483647 + 10L << endl;
    cout << "long(2147483647 + 10) = " << long(2147483647 + 10) << endl;
    cout << "long(2147483647) + 10 = " << long(2147483647) + 10 << endl;

    return 0;  // success
}
```

```
'a'              = a
char(97)         = a
'a'+5            = 102
char(97)+5       = 102
char('a'+5)      = f
char(char(97)+5) = f
char(97)+char(5) = 102
```

```
int('a')     = 97
double('a') = 97
int(97)     = 97
double(97)  = 97
int(5.7)    = 5
double(5.7) = 5.7
char(98)    = b
char(98.6)  = b

1/2                    = 0
1/2.0                  = 0.5
2147483647 + 10        = -2147483639
2147483647 + 10L       = 2147483657
long(2147483647 + 10) = -2147483639
long(2147483647) + 10 = 2147483657
```

Things to Think About

- What the heck do `<<` and `>>` do in the context of input and output streams? (Note: Looking into this thoroughly will take you into the realm of objects, operator overloading, and all sorts of fun stuff you won't be seeing in class for a long while. Still worth looking into though.)

  `http://www.cplusplus.com/reference/ostream/ostream/operator%3C%3C/`

  `http://www.cplusplus.com/reference/istream/istream/operator%3E%3E/`

- What's a "stream"?

  `http://stackoverflow.com/questions/12145357/c-what-is-a-stream`

  `http://www.cprogramming.com/tutorial/c++-iostreams.html`

- Why `static_cast` instead of other types of casting?

  `http://stackoverflow.com/questions/103512/in-c-why-use-static-castintx-instead-of-i`

- What is a `namespace`, and why are they important?

  `http://www.cprogramming.com/tutorial/namespaces.html`

Operator Precedence and Associativity List

From `http://www.cplusplus.com/doc/tutorial/operators/` near the bottom of the page.

From greatest to smallest priority, C++ operators are evaluated in the following order:

Level	Precedence group	Operator	Description	Grouping
1	Scope	`::`	scope qualifier	Left-to-right
2	Postfix (unary)	`++ --`	postfix increment / decrement	Left-to-right
		`()`	functional forms	
		`[]`	subscript	
		`. ->`	member access	
3	Prefix (unary)	`++ --`	prefix increment / decrement	Right-to-left
		`~ !`	bitwise NOT / logical NOT	
		`+ -`	unary prefix	
		`& *`	reference / dereference	
		`new delete`	allocation / deallocation	
		`sizeof`	parameter pack	
		`(`*`type`*`)`	C-style type-casting	
4	Pointer-to-member	`.* ->*`	access pointer	Left-to-right
5	Arithmetic: scaling	`* / %`	multiply, divide, modulo	Left-to-right
6	Arithmetic: addition	`+ -`	addition, subtraction	Left-to-right
7	Bitwise shift	`<< >>`	shift left, shift right	Left-to-right
8	Relational	`< > <= >=`	comparison operators	Left-to-right
9	Equality	`== !=`	equality / inequality	Left-to-right
10	And	`&`	bitwise AND	Left-to-right
11	Exclusive or	`^`	bitwise XOR	Left-to-right
12	Inclusive or	`|`	bitwise OR	Left-to-right
13	Conjunction	`&&`	logical AND	Left-to-right
14	Disjunction	`||`	logical OR	Left-to-right
15	Assignment-level expressions	`= *= /= %= += -= >>= <<= &= ^= |=`	assignment / compound assignment	Right-to-left
		`?:`	conditional operator	
16	Sequencing	`,`	comma separator	Left-to-right

When an expression has two operators with the same precedence level, *grouping* determines which one is evaluated first: either left-to-right or right-to-left.