Review Ch 1-3
Executing code

Compile code
(convert from C++ to computer code)
- Syntax errors will prevent compilation

Run code
- Runtime errors will crash your program
- Logic errors will make your program give the wrong output
Syntax =
car won't start

Runtime =
car accident

Logic =
bad directions
Identifiers

The identifier is the name of a variable/method
- Case sensitive
- Must use only letters, numbers or _
- Cannot start with a number
- (Some reserved identifiers)

Examples (second word):
  int x, String s_o_s, double high2low
Primitive Types

bool - True or false
char - (character) A letter or number
int - (integer) Whole numbers
long - (long integers) Larger whole numbers
float - Decimal numbers
double - Larger decimal numbers

doubles are approximations
ints are exact but have a more limited range
cin

```cpp
cin >> x;
By default, this will read the based off the
type of x, until it finds a space or character
not the same type as x
```

```cpp
getline(cin, x);
x needs to be a string, but then stores
everything up until you hit enter
```

Note: mixing getline and “cin >>” ends poorly
Operations

Order of precedence (higher operations first):
- -, +, ++, -- and ! (unary operators)
* *, / and % (binary operators)
+ + and - (binary operators)

Operators that change variables:
++ , -- , += , -= , *= , /= , =

Note: integer division happens if you divide two ints: int / int = int
If statements

```java
if (boolean expression) {
    // code
}
else {
    // more code
}

|| is the OR operations
&& is the AND operations
```

Logical operations:

- `>` (greater than)
- `==` (equals)
- `<` (less than)
- `>=` (greater than or equal to)
- `!=` (not equal to)
- `<=` (less than or equal to)
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
Scope

Variables only exist in the most recently started block:

```c
if(x < y)
{
    int z = 9;
}
```

If you want variables to exist longer, you need to declare them further up in the program.

z lives in most recent block
z goes away at corresponding closing block
Loops

3 parts to any (good) loop:
- Loop variable initialized
- boolean expression with loop variable
- Loop variable updated inside loop

for loops have these 3 parts in the same place
while loops have these spread out
do while loops are while loops that always execute at least once
Looping control commands

`continue` restarts loop immediately

```java
for (i = 0; i < 10; i++)
{
    // code will run everytime
    if (doSkip)
    {
        continue;
    }
    // code will not run
    // if doSkip is true
}
```

`break` stops loop

```java
for (i = 0; i < 10; i++)
{
    // code
    // code will run everytime
    if (doSkip)
    {
        break;
    }
}
// outside loop code
```