Signed and Unsigned

Unlike Java, C++ integers come in two "flavors": signed and unsigned. Unsigned variables offer **twice the range of positive numbers**, but cannot store negative numbers. For example, a 32-bit `int` has a maximum value of 2,147,483,647, while the maximum `unsigned int` is 4,294,967,295. C++ allows `unsigned int` to be abbreviated as `unsigned`.

Since integers use a fixed amount of memory, what happens if you exceed their range? Unsigned numbers will "wrap around". For instance, try this.

```
unsigned n = 0;
cout << n - 1 << endl;
```

As you can see, the output wraps around from zero to the largest possible `unsigned` value.

```
4294967295
```

Signed Overflow

This is not necessarily the case with signed numbers, however. Overflow and underflow on signed numbers is **undefined behavior**. Consider this code:

```
int n = 2147483647; // max size of 32-bit int
cout << "one larger is " << n + 1 << endl;
```

On many modern compilers (including ours), if you add the compiler flag `-fsanitize=undefined`, you will get a runtime error, like that shown here.

```
overflow.cpp:7:37: runtime error: signed integer overflow:
2147483647 + 1 cannot be represented in type 'int'
```

If you leave off that flag, most compilers wrap around just as with signed numbers, and it will print this:

```
one larger is -2147483648
```