# Concatenation & Comparison

**The `<string>` library redefines several standard operators using a C++ feature** called **operator overloading**. When you use the `+` operator with numbers, it means addition, but, when you use it with the `string` type, it means **concatenation**.

```
string s1 = "hello", s2 = "world";
string s2 = s1 + " " + s2;          // "hello world"
```

The shorthand `+=` assignment operator has also been overloaded. It concatenates new text to the end of an existing `string`. You may concatenate `char` values to a `string` object, but you **cannot** concatenate numbers to `string` objectss as you could in Java.

```
string s{"abc"};  // uniform initialization
s += s;           // ok, "abcabc"
s += "def";       // literal ok, "abcabcdef"
s += 'g';         // char ok, "abcabcdefg"
s = s + 2;        // ERROR; no conversion
```

You **cannot** concatenate two string literals: `"a" + "b"` is **illegal**. However, separating them with whitespace, like `"a" "b"`, is legal. Use this is used to join long lines together.

## Comparisons

C++ overloads the **relational operators** so that you can **compare** `string` values just like primitive types. To see if the value of `str` is equal to `"quit"`, just write this:

```
if (str == "quit") . . .
```

There is no need to use `equals()` or `compareTo()` as in Java.

Strings are compared using **lexicographic ordering**. Informally that means a `string` is smaller if it would appear earlier in the dictionary. However, when doing comparisons, case is significant, so `"abc"` is **not** equal to `"ABC"`. Upper-case characters are "smaller" than lower-case characters, because they have smaller ASCII values.