Range-based Loops

As mentioned earlier, range-based loops were added in C++11. A range loop will visit each element in a collection, setting the range variable to each value in the collection, in turn. These loops are very similar to the simplified `for` loop added in Java 5, and the `for in` loops in Python.

Let's look at the three variations of range-based loops which are offered in C++. We'll start with **value iteration**, which is the closest to the version used in Java.

Here's a short example, which prints each character in a string on a line by itself:

```
string snake{"Ouroboros"};
for (char c : snake)
{
    cout << c << endl;
}
```

On each loop cycle, the variable `c` is initialized with a **copy** of the next character in the `string snake`. When all of the characters have been processed, the loop stops. Thus, you can read this loop as saying *"for each character in snake, do something"*.

With value iteration, changes to the variable `c` have no effect on the `string snake`. If you **want** to change the `string` itself, then use **reference iteration**. Here's a second example which does that:

```
string str{"one two three"};
for (char& c : str)
{
    if (c == ' ') { c == '_'; }
}
cout << str << endl;
```

Finally, if the items you are iterating over are very large, and you want to make sure you don't change them, then you'd use **const-ref iteration** like this (made up) example.

```
Album photos(get_phone_photos());
for (const Image& img : photos)
{
    if (is_cute_cat(img))
    {
        display(img);
    }
}
```

Because each picture in your photos library might be very large, you **wouldn't** want to copy them with value iteration. Similarly, since you wouldn't want the **CuteCats** app to have the ability to modify (or perhaps erase) your cute cat photos, the loop should access each variable as a `const Image&`.