String Reference Parameters

Since reference parameters don't make a copy of the argument, they are much more efficient when passing a class-type argument such as **`string`** or **`vector`**. What if you were to change the heading of **`count_vowels`** like this. Would that work?

```
int count_vowels(string& str)
```

Well, yes and no!

- Because the parameter **`str`** is now a **reference**, there is no copy made, so it is **much more efficient**.
- However, because it is a reference, you can now only call the **`count_vowels`** function with an ***lvalue***. You could no longer write: `count_vowels("hello");`. Your function is much less **usable**.
- Finally, since **`str`** is a reference, there is nothing to **prevent** the **`count_vowels`** function from **inadvertently modifying** the parameter, and, thus by extension, the argument. The function is not as **safe** as it could be.

The solution is simple, however. **Whenever** you pass a **`string`** as an argument to a function, use **`const string&`** for the parameter if the function **will not** modify the calling argument, and **`string&`** if it will.

Here is the improved header for **`count_vowels`**, which is correct, efficient and safe.

```
int count_vowels(const string& str)
```

*If the **`string`** **should** be modified use a regular reference. If the string **should not** be modified, use a **`const`** reference as your parameter type.*

You can add these C++11 **type alias declarations** to your programs to make this easier if you like:

```
using stringIn = const string&;   // input string not modified
using stringRef = string&         // output string, modified
```