Potential Pitfalls

Surprising number of template instantiations with string literals

When forwarding references are used as a means to avoid code repetition between exactly two overloads of the same function (one accepting a `const T&` and the other a `T&&`), it can be surprising to see more than two template instantiations for that particular template function, in particular when the function is invoked using **string literals**.

Consider, as an example, a `Dictionary` class containing two overloads of an `addWord` member function:

```
class Dictionary
{
    // ...

public:
    void addWord(const std::string& word);  // (0) copy word in the dictionary
    void addWord(std::string&& word);       // (1) move word in the dictionary
};

void f()
{
    Dictionary d;

    std::string s = "car";
    d.addWord(s);                    // invokes (0)

    const std::string cs = "toy";
    d.addWord(cs);                   // invokes (0)

    d.addWord("house");              // invokes (1)
    d.addWord("garage");             // invokes (1)
    d.addWord(std::string{"ball"});  // invokes (1)
}
```

Now, imagine replacing the two overloads of `addWord` with a single *perfectly forwarding* template member function, with the intention of avoiding code repetition between the two overloads:

```
class Dictionary
{
    // ...

public:
    template <typename T>
    void addWord(T&& word);
};
```

Perhaps surprisingly, the number of template instantiations skyrockets: