Templated Variable Declarations/Definitions

Traditional **`template`** syntax is extended to define, in namespace or class (but not function) scope, a family of like-named variables ~~that~~ can be instantiated ~~explicitly~~.

Description

By beginning a variable declaration with the familiar **`template-head`** syntax (e.g., **`template`** `<`**`typename`** `T>`), we can create a *variable template*, which defines a family of variables having the same name (e.g., `exampleOf`):

```
template <typename T> T exampleOf;  // variable template defined at file scope
```

Like any other kind of template, ~~a variable template can be instantiated explicitly~~ by providing an appropriate number of ~~type or non-type~~ arguments:

```
#include <iostream>  // std::cout

void initializeExampleValues()
{
    exampleOf<int>   = -1;
    exampleOf<char>  = 'a';
    exampleOf<float> = 12.3f;
}

void printExampleValues()
{
    initializeExampleValues();
    std::cout << "int = "   << exampleOf<int>   << "; "
              << "char = "  << exampleOf<char>  << "; "
              << "float = " << exampleOf<float> << ';';

    // outputs "int = -1; char = a; float = 12.3;"
}
```

In the example above, the type of each instantiated variable is the same as its template parameter, but this matching is not required. For example, the type might be the same for all instantiated variables or derived from its parameters, such as by adding **`const`** qualification: