Type/Template Aliases (Extended typedef)

The **using** keyword may now be used to introduce type aliases and alias templates, providing a more general alternative to **typedef** that might also improve readability, especially for function aliases.

Description

The keyword **using** has historically supported the introduction of an alias for a named entity, e.g., type, function, or data, from some named scope into the current one. As of C++11, we can employ the **using** keyword to achieve everything that could previously be accomplished with a **typedef** declaration but in a syntactic form that many people find more natural and intuitive but that offers nothing profoundly new:

```
using Type1 = int;     // equivalent to typedef int Type1;
using Type2 = double;  // equivalent to typedef double Type2;
```

In contrast to **typedef**, the name of the synonym created via the **using** syntax always appears on the left side of the = token and separate from the type declaration itself, the advantage of which becomes apparent with more involved types, such as *pointer-to-function*, *pointer-to-member-function*, or *pointer-to-data-member*:

```
struct S { int i; void f(); };  // user-defined type S defined at file scope

using Type3 = void(*)();        // equivalent to typedef void(*Type3)();
using Type4 = void(S::*)();     // equivalent to typedef void(S::*Type4)();
using Type5 = int S::*;         // equivalent to typedef int S::*Type5;
```

Just as with a **typedef**, the name representing the type can be qualified, but the symbol representing the synonym cannot:

```
namespace N { struct S { }; }  // original type S defined with namespace N

using Type6 = N::S;            // equivalent to typedef N::S Type6;
using ::Type7 = int;           // Error, the alias's name must be unqualified.
```

Unlike a **typedef**, however, a type alias introduced via **using** can itself be a template, known as an *alias template*:

```
template <typename T>
using Type8 = T;  // "identity" alias template

Type8<int>    i;  // equivalent to int i;
Type8<double> d;  // equivalent to double d;
```