Compile-Time Invocable Functions

Functions decorated with **constexpr** are eligible to be invoked as part of a **constant expression**.

Description

A constant expression is an expression whose value is determined at compile time — i.e., one that could be used, say, to define the size of a C-style array or as the argument to a **static_assert**:

```
enum { e_SIZE = 5 };             // e_SIZE is a constant expression of value 5.
int a[e_SIZE];                   // e_SIZE must be a constant expression.
static_assert(e_SIZE == 5, "");  //    "     "   " "     "         "
```

Prior to C++11, evaluating a conventional function at compile time as part of a constant expression was not possible:

```
inline const int z() { return 5; }  // OK, returns a nonconstant expression
int a[z()];                         // Error, z() is not a constant expression.
static_assert(z() == 5, "");        // Error,  "   "  "  "     "         "
int a[0 ? z() : 9];                 // Error,  "   "  "  "     "         "
```

Developers, in need of such functionality, would use other means, such as template metaprogramming, external code generators, preprocessor macros, or hard-coded constants (as shown in the example above), to work around this deficiency.

As an example, consider a **metaprogram** to calculate the nth factorial number:

```
template <int N>
struct Factorial { enum { value = N * Factorial<N-1>::value }; };  // recursive

template <>
struct Factorial<0> { enum { value = 1 }; };  // base case
```

Evaluating the `Factorial` metafunction in the example above on a constant expression results in a constant expression:

```
static_assert(Factorial<5>::value == 120, "");  // OK
int a[Factorial<5>::value];                     // OK, array of 120 ints
```

Note, however, that the metafunction can be used only with template arguments that themselves must be constant expressions:

```
int factorial(const int n)
{
    static_assert(n >= 0, "");    // Error, n is not a constant expression.
    return Factorial<n>::value;   // Error, "  "  "  "     "         "
}
```