Lambdas Having a Templated Call Operator

C++14 extends the **lambda expression** syntax of C++11 to allow a *templated* ~~definition~~ of the function-call operator belonging to the **closure type**.

Description

Generic lambdas are a C++14 extension of C++11 lambda expressions (see Section 2.1. "Lambdas" on page 573) for which the function-call operator is a member function template, which enables the deduction of template argument types at the point of invocation.

Consider two lambda expressions, each of which simply returns its argument:

```
auto identityInt = [](int  a) { return a; };  // nongeneric lambda
auto identity =    [](auto a) { return a; };  // generic lambda
```

Generic lambdas are characterized by the presence of one or more **auto** parameters, accepting arguments of any type. In the example above, the first version is a nongeneric lambda having a parameter of concrete type **int**. The second version is a generic lambda because its parameter uses the placeholder type **auto**. Unlike `identityInt`, which is callable only for arguments implicitly convertible to **int**, `identity` can be applied to any type that can be passed by value:

```
int         a1 = identityInt(42);    // OK, a1 == 42
double      a2 = identityInt(3.14);  // Bug, a2 == 3, truncation warning
const char* a3 = identityInt("hi");  // Error, cannot pass "hi" as int
int         a4 = identity(42);       // OK, a4 == 42
double      a5 = identity(3.14);     // OK, a5 == 3.14
const char* a6 = identity("hi");     // OK, strcmp(a6, "hi") == 0
```

Generic lambdas accomplish this compile-time polymorphism by ~~defining~~ their function-call operator — **operator()** — as a *template*. Recall that the result of a lambda expression is a **closure object**, an object of unique type having a function-call operator; i.e., the closure type is a unique **functor** class. The parameters ~~defined in~~ the lambda expression become the parameters to the function-call operator. The following code transformation is roughly equivalent to the definitions of the `identityInt` and `identity` closure objects from the example above:

```
struct __lambda_1  // compiler-generated name; not visible to the user
{
    int operator()(int x) const { return x; }
    // ...
};
```