Using = delete for Arbitrary Functions

~~The keyword **delete** annotating a function's first declaration~~ makes any attempt to use or even access it ill formed.

Description

Declaring a particular function ~~or function overload~~ to result in a fatal diagnostic upon invocation can be useful, e.g., to suppress the generation of a **special member function** or to limit the types of arguments a particular overload set is able to accept. In such cases, **= delete** followed by a semicolon (;) can be used in place of the body of any function on first declaration only to force a compile-time error if any attempt is made to invoke it or take its address.

```
void g(double) { }
void g(int) = delete;

void f()
{
    g(3.14);  // OK, f(double) is invoked.
    g(0);     // Error, f(int) is deleted.
}
```

Notice that deleted functions participate in **overload resolution** and produce a compile-time error when selected as the best candidate.

Use Cases

Suppressing special member function generation

When instantiating an object of user-defined type, special member functions that have not been declared explicitly are often generated automatically by the compiler. The generation of individual special member functions can be affected by the existence of other user-defined special member functions or by limitations imposed by the specific types of any data members or base types; see Section 1.1."Defaulted Functions" on page 33. For certain kinds of types, the notion of *copying* is not meaningful, and hence permitting the compiler to generate *copy* operations would be inappropriate. The two special member functions controlling **move operations**, introduced in C++11, are typically implemented as effective optimizations of **copy operations** and thus would be similarly contraindicated. Much less frequently, a useful notion of moving exists where copying does not, and so we might choose to have move operations generated, while copy operations are explicitly deleted; see Section 2.1. "*Rvalue* References" on page 710.