```
struct V
{
    int v;
              operator    int() const { return v; }  // implicit conversion
    constexpr operator double() const { return v; }  // implicit conversion
};

struct S
{
    int i; double d;  // A constexpr constructor must initialize both members.

    constexpr S(const V& x, double y) : i(x), d(y) { }  // Error, the needed
        // int implicit conversion is not declared constexpr.

    constexpr S(int x, const V& y) : i(x), d(y) { }    // OK, the needed
        // double implicit conversion is declared constexpr.
};
```

constexpr function templates

Function templates, member function templates, and constructor templates can all be declared **constexpr** ~~and~~ more liberally than nontemplated entities. That is, if a particular instantiation of such a template doesn't meet the requirements of a **constexpr** function, member function, or constructor, it will not be invocable at compile time.[8] For example, consider a function template, `sizeOf`, that can be evaluated at compile time only if its argument type, `T`, is a literal type:

```
template <typename T> constexpr int sizeOf(T t) { return sizeof(t); }
    // This function is constexpr only if T is a literal type.

struct S0 { int i;           S0() : i(0) { } };  // not a literal type
struct S1 { int i; constexpr S1() : i(0) { } };  // a literal type

int a[sizeOf(int())];  // OK,    int is     a literal type.
int b[sizeOf( S0())];  // Error,  S0 is not a literal type.
int c[sizeOf( S1())];  // OK,     S1 is     a literal type.
```

If no specialization of such a function template would yield a **constexpr** function, then the program is IFNDR. For example, if this same function template were implemented in C++11 with a function body consisting of more than just a single **return** statement, it would be ill formed:

[8] A specialization that cannot be evaluated at compile time is, however, still considered **constexpr**. This idiosyncracy is not readily observable but does enable some generic code to remain well formed as long as the particular specializations are not actually required to be evaluated at compile time. This rule was adopted with the resolution of CWG issue 1358 (**smith11a**).