Initially, this function template will support just two built-in types, **`float`** and **`double`**, but it is anticipated to eventually support the additional built-in type **`long double`** and perhaps even supplementary user-defined types (e.g., `Float128`) to be made available via separate headers (e.g., `float128.h`). ~~By placing~~ only the declaration of the `transform` function template in its component's header, clients will be able to link against only two supported explicit specializations provided in the `transform.cpp` file:

```
// transform.cpp:
#include <transform.h>  // Ensure consistency with client-facing declaration.

template <typename T>   // redeclaration/definition of free-function template
T transform(const T& value)
{
    // insulated implementation of transform function template
}

// explicit-instantiation definitions
template float transform(const float&);    // Instantiate for type float.
template double transform(const double&);  // Instantiate for type double.
```

Without the two explicit-instantiation declarations in the `transform.cpp` file above, its corresponding object file, `transform.o`, would be empty.

Note that, as of C++11, we *could* place the corresponding explicit-instantiation declarations in the header file for, say, documentation purposes:

```
// transform.h:
#ifndef INCLUDED_TRANSFORM
#define INCLUDED_TRANSFORM

template <typename T>  // declaration only of free-function template
T transform(const T& value);
    // Return the transform of the specified floating-point value.

// explicit-instantiation declarations, available as of C++11
extern template float transform(const float&);    // user documentation only;
extern template double transform(const double&);  // has no effect whatsoever

#endif
```

Because no definition of the `transform` free-function template is visible in the header, no *implicit* instantiation can result from client use; hence, the two explicit-instantiation declarations above for **`float`** and **`double`**, respectively, do nothing.