Variadic function templates

Let's first consider a variadic function template that accepts a template parameter pack in its template parameter list but does not use any of its template parameters in its **function parameter list**:

```
template <typename... Ts>
int f0a();     // does not use Ts in parameter list

template <typename... Ts>
int f0b(int);  // uses int but not Ts in parameter list

template <typename T, class... Ts>
int f0c();     // does not use T or Ts in parameter list
```

The only way to call the functions shown in the code snippet above is by using explicit template argument lists:

```
int a1 = f0a<int, char, int>();  // Ts=<int, char, int>
int a2 = f0b<double, void>(42);  // Ts=<double, void>
int a3 = f0c<int, void, int>();  // T=int, Ts=<void, int>
int e1 = f0a();                  // Error, cannot deduce Ts
int e2 = f0b(42);                // Error, cannot deduce Ts
int e3 = f0c();                  // Error, cannot deduce T and Ts
```

The notation `f0a<int, char, int>()` means to explicitly instantiate template function `f0a` with the specified type arguments and to call that instantiation.

Invoking any of the instantiations shown above will, of course, require that the corresponding definition of the function template exist somewhere within the program:

```
template <typename...> void f0a() { /*...*/ }  // variadic template definition
```

This definition will typically be part of or reside alongside its declaration in the same header or source file, but see Section 2.1."**`extern template`**" on page 353.

With the notable exception of **factory functions** such as `make_shared`, such functions are rarely encountered in practice; most of the time a template function would use its template parameters in the function parameter list. Let's now consider a different kind of variadic function template — one that accepts an arbitrary number of template arguments and an arbitrary number of *function* arguments working in tandem with the template arguments.

Function parameter packs

The syntax for declaring a variadic function template that accepts an arbitrary number of function arguments makes two distinct uses of the ellipsis (`...`) token. The first use is to introduce the template parameter pack `Ts`, as already shown. Then, to make the function parameter list of a function template variadic, we introduce a function parameter pack by placing the `...` to the left of the function parameter name: