【C++】Day17

Class	C++
Date	@December 7, 2021
Material	
Series Number	
Summary	Functions with Varying Parameter

【Ch6】Functions with Varyinig Parameters

6.2.6 Functions with Varying Parameters

The new standard provides two primary ways to write a function that takes a varying number of arguments:

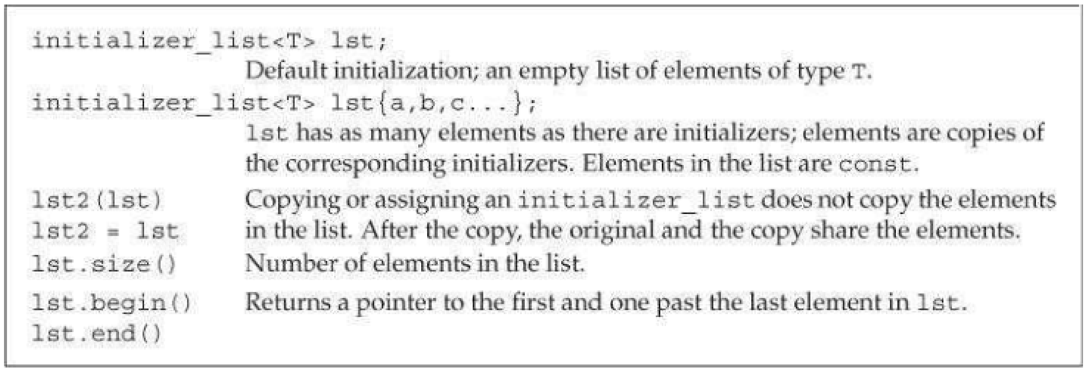
1. If all the arguments have the same type, we can pass a library type named initialier_list.
2. If the argument types vary, we can write a special kind of function, known as a variadic template.

initializer_list Parameters

We can write a function that takes an unknown number of arguments of a single type by using an `initializer_list` parameter.

An initializer_list is a library type that represents an array of values of the specified type. This type is defined in the initializer_list header.

Table 6.1. Operations on initializer_lists



`initializer_list<T> lst;`	Default initialization; an empty list of elements of type `T`.
`initializer_list<T> lst{a,b,c...};`	`lst` has as many elements as there are initializers; elements are copies of the corresponding initializers. Elements in the list are `const`.

`lst2(lst)`
`lst2 = lst`
	Copying or assigning an `initializer_list` does not copy the elements in the list. After the copy, the original and the copy share the elements.

`lst.size()`	Number of elements in the list.

`lst.begin()`
`lst.end()`
	Returns a pointer to the first and one past the last element in `lst`.

Like a vector, initializer_list is a template. When we define an initializer_list, we must specify the type of the elements that the list will contain:

```
initializer_list<string> ls; //initializer_list of strings
initializer_list<int> li;
```

Unlike vector, the elements in an initializer_list are always const values;o there is no way to change the value of an element in an initializer_list;

We can write our function to produce error messages from a varying numboer of arguments as follows:

```
void error_msg(initializer_list<string> il) {
  for(auto beg = il.begin(); beg != il.end(); ++beg)
    cout << *beg << " ";
  cout << endl;
}
```

The begin and end operations on initializer_list objects are analogous to the corresponding vector members.

When we pass a sequence of values to an initializer_list parameter, we must enclose the sequence in curly braces:

```
if (expected != actual)
  error_msg({"functionX", expected, unexpected});
else
  error_msg({"functionX", "ok"});
```