The symbol `T` is called a *type parameter.* It is simply a place holder that is replaced by an actual type or class when the function is invoked.

A function template is declared the same way as an ordinary function, except that it is preceded by the specification

```
template <class T>
```

and the type parameter `T` may be used in place of ordinary types within the function definition. The use of the word `class` here means "any type." More generally, a template may have several type parameters, specified like this:

```
template <class T, class U, class V>
```

Function templates are called the same way ordinary functions are called:

```
int m = 22, n = 66;
swap(m, n);
string s1 = "John Adams", s2 = "James Madison";
swap(s1, s2);
Rational x(22/7), y(-3);
swap(x, y);
```

For each call, the compiler generates the complete function, replacing the type parameter with the type or class to which the arguments belong. So the call `swap(m,n)` generates the integer `swap` function shown above, and the call `swap(s1, s2)` generates the `swap` function for `string` objects.

Function templates are a direct generalization of function overloading. We could have written several overloaded versions of the `swap` function, one for each type that we thought we might need. The single `swap` function template serves the same purpose. But it is an improvement in two ways. It only has to be written once to cover all the different types that might be used with it. And we don't have to decide in advance which types we will use with it; any type or class can be substituted for the type parameter `T`. Function templates share source code among structurally similar families of functions.

EXAMPLE 13.2 The Bubble Sort Template

This is the Bubble Sort (Example 6.13 on page 134) and a print function for vectors of any base type.

```
template<class T>
void sort(T* v, int n)
{ for (int i = 1; i < n; i++)
    for (int j = 0; j < n-i; j++)
      if (v[j] > v[j+1]) swap(v[j], v[j+1]);
}

template<class T>
void print(T* v, int n)
{ for (int i = 0; i < n; i++)
  cout << " " << v[i];
  cout << endl;
}

int main()
{ short a[9] = {55, 33, 88, 11, 44, 99, 77, 22, 66};
```