Function Templates

I hope the code on the preceding page bothers you as much as it bothers me. It doesn't take much to notice that **the body of each function is identical**. Why can't we define one version of the function that takes any kind of argument?

Surprise! We can!

C++ functions with **generic types** are called **function templates**. (In Java these are called generic functions, but **template** is used more often in C++).

You define a function template with the same syntax as a regular function, **preceded by** the `template` keyword and a series of **template parameters** inside angle-brackets `<>`.

```
1  template <typename T>
2  std::string to_string(const T& n)
3  {
4      std::ostringstream out;
5      out << n;
6      return out.str();
7  }
```

The template parameters are separated by commas, and use **generic template type names**: names preceded by either the `class` or `typename` keyword followed by an identifier. Both keywords **are synonyms** in template declarations

When using separate compilation:

- Function templates ar **placed inside the header file**, unlike normal functions, which are placed inside the implementation file.
- You must **fully qualify all library types**, such as `string` and `ostringstream` in the example shown here, since you are not allowed to add `using namespace std;` to a header file.