Chapter 6 introduces templates, which are a C++ feature that easily permits the reuse of existing code for new purposes.

This presentation shows how to implement and use the simplest kinds of templates: template functions.
Here’s a small function that you might write to find the maximum of two integers.

```plaintext
int maximum(int a, int b)
{
    if (a > b)
        return a;
    else
        return b;
}
```
Here’s a small function that you might write to find the maximum of two double numbers.

```c
int maximum(double a, double b) {
    if (a > b) {
        return a;
    } else {
        return b;
    }
}
```
Finding the Maximum of Two Knafs

Here’s a small function that you might write to find the maximum of two Knafs.

```c
int maximum(Knafn a, Knafn b) {
    if (a > b) {
        return a;
    } else {
        return b;
    }
}
```
Suppose your program uses 100,000,000 different data types, and you need a maximum function for each...
This template function can be used with many data types.

```cpp
template <class Item>
Item maximum(Item a, Item b)
{
    if (a > b)
        return a;
    else
        return b;
}
```
When you write a template function, you choose a data type for the function to depend upon...

```cpp
template <class Item>
Item maximum(Item a, Item b)
{
    if (a > b)
        return a;
    else
        return b;
}
```
A template prefix is also needed immediately before the function’s implementation:

```cpp
template <class Item>
Item maximum(Item a, Item b)
{
    if (a > b)
        return a;
    else
        return b;
}
```
Once a template function is defined, it may be used with any adequate data type in your program...

```cpp
template <class Item>
Item maximum(Item a, Item b)
{
    if (a > b)
        return a;
    else
        return b;
}
```

```
cout << maximum(1, 2);
cout << maximum(1.3, 0.9);
...
```
Here’s another function that can be made more general by changing it to a template function:

```c
int array_max(int data[ ], size_t n)
{
    size_t i;
    int answer;

    assert(n > 0);
    answer = data[0];
    for (i = 1; i < n; i++)
        if (data[i] > answer) answer = data[i];
    return answer;
}
```
Here’s the template function:

```cpp
template <class Item>
Item array_max(Item data[], size_t n)
{
    size_t i;
    Item answer;

    assert(n > 0);
    answer = data[0];
    for (i = 1; i < n; i++)
        if (data[i] > answer) answer = data[i];
    return answer;
}
```
A template function depends on an underlying data type.

More complex template functions and template classes are discussed in Chapter 6.
Presentation copyright 1997, Addison Wesley Longman, 
For use with *Data Structures and Other Objects Using C++* 
by Michael Main and Walter Savitch.

Some artwork in the presentation is used with permission from Presentation Task Force 
(copyright New Vision Technologies Inc) and Corel Gallery Clipart Catalog (copyright 
Corel Corporation, 3G Graphics Inc, Archive Arts, Cartesia Software, Image Club 

Students and instructors who use *Data Structures and Other Objects Using C++* are welcome 
to use this presentation however they see fit, so long as this copyright notice remains 
intact.