

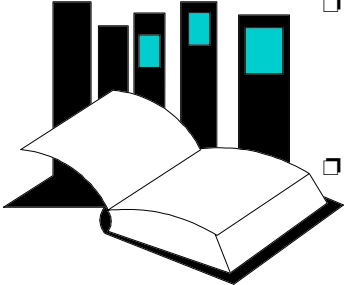


# Template Functions



- ❑ 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.



**CHAPTER 6**  
Data Structures and Other Objects

Templates are an important part of C++ that allows a programmer to reuse existing code for new purposes. In some sense, templates ensure that you don't have to continually "reinvent the wheel."

This lecture introduces how to implement and use template functions. The best time for the lecture is just before the students read Chapter 6.

## Finding the Maximum of Two Integers

- Here's a small function that you might write to find the maximum of two integers.

```
int maximum(int a, int b)
{
    if (a > b)
        return a;
    else
        return b;
}
```

To illustrate template functions, let's look at this simple function. It would appear as part of a program that needs to find the larger of two integers. The function returns a copy of the larger of its two arguments.

## Finding the Maximum of Two Doubles

- Here's a small function that you might write to find the maximum of two double numbers.

```
int maximum(double a, double b)
{
    if (a > b)
        return a;
    else
        return b;
}
```

Suppose your program also needs to find the larger of two double numbers. Then you could add a second function for double numbers. By the way, you may use the same name, `maximum`, for the two functions. The compiler is smart enough to determine which function your program is using at any given point, by looking at the types of the parameters.

## Finding the Maximum of Two Knafns

- Here's a small function that you might write to find the maximum of two Knafns.

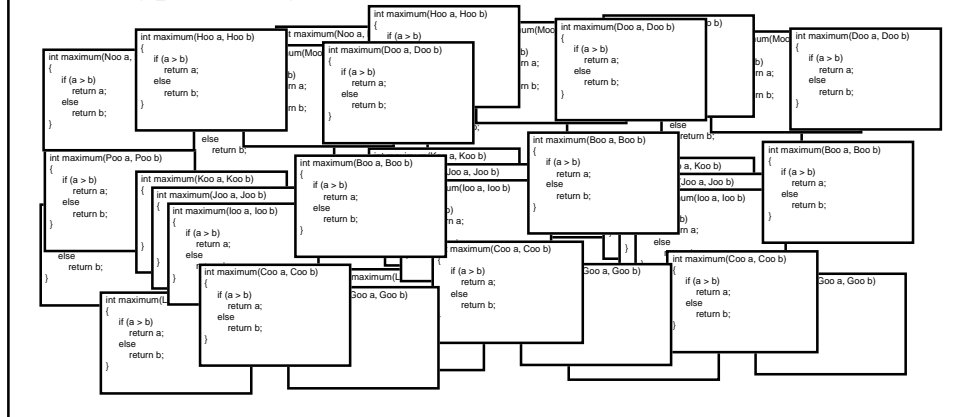
```
int maximum(Knafn a, Knafn b)
{
    if (a > b)
        return a;
    else
        return b;
}
```

Suppose your program also has another data type, called Knafn. This could be a class that you wrote yourself for some purpose.

Anyway, perhaps Knafn objects can be compared with the > operation (because the Knafn implementation includes an overloaded > operator). In this case, your program might have a third function to compare two Knafn objects.

## One Hundred Million Functions...

- Suppose your program uses 100,000,000 different data types, and you need a maximum function for each...



In fact, for your next programming assignment, I'm going to give you a printed list of 100,000,000 different data types, each of which has the `>` operator defined. Your job will be to implement a maximum function for each of the types. How long do you think this job will take?

How many of you think it will take more than one day's work? Raise your hands.

Okay, now how many of you think you'll need less than a day--in fact less than five minutes? Raise your hands. These are the people that have been reading ahead in Chapter 6 about a feature called template functions.

## A Template Function for Maximum

- This template function can be used with many data types.

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

This is our first example of a template function. It is a single function that depends on an underlying data type that I've called `Item`. The actual `Item` type could be any of many different types: `int`, `double`, `char--` in fact any type that has two features: (1) the data values can be compared with the `>` operator, and (2) the data type has a copy constructor. (By the way, where is the copy constructor needed? It is needed to initialize the two parameters from their actual arguments, and also needed when the function returns a value.)

Anyway, let's look at the pattern for converting an ordinary function to a template function...

## A Template Function for Maximum

- When you write a template function, you choose a data type for the function to depend upon...

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

Our maximum function depends on the type of the item that we are comparing. This is the type of the two parameters, and also the return type. So, in the template function, we choose a name for this type--we use the name `Item`--and you use that name instead of the actual data type name.

## A Template Function for Maximum

- A template prefix is also needed immediately before the function's implementation:

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

Before an implementation, a special template prefix is also required. The prefix consists of the keyword `template` followed by angle brackets. Inside the angle brackets is the keyword `class` and the name that you choose to describe the underlying data type.



## Using a Template Function

- Once a template function is defined, it may be used with any adequate data type in your program...

```
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 are two examples of how the single template function can be used in two different ways in a program. In the first example, the compiler sees two integer arguments, so the compiler will automatically create a version of the function where the `Item` is an `int`. In the second example, the compiler sees two double arguments, so the compiler will automatically create a version of the function where the `Item` is a `double`. We could have a third example where the two arguments were `Knafn` objects, and the compiler would automatically create a version of the function where the `Item` is a `Knafn`.

## Finding the Maximum Item in an Array

- Here's another function that can be made more general by changing it to a template function:

```
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 another function that can be improved by making it into a template function. In this form, the function can find the largest integer in an array of integers. But it can't be used for an array of double numbers, or an array of some other kind of objects.

In this case, the function depends on the underlying data type of the components of the array. We can call this component type `Item`, and rewrite the function as a template function...

## Finding the Maximum Item in an Array

□ Here's the template function:

```
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;
}
```

Here's the template version. Notice these things:

1. the template prefix
2. The change of the data type of at least one parameter (the array)
3. The change of the return type (this might not occur for every template function)
4. In the implementation, we can use the Item type for local variables



## Summary

- ❑ A template function depends on an underlying data type.
- ❑ More complex template functions and template classes are discussed in Chapter 6.

A quick summary . . .

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  
Graphics Inc, One Mile Up Inc, TechPool Studios, Totem Graphics Inc).

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.



Feel free to send your ideas to:

Michael Main

[main@colorado.edu](mailto:main@colorado.edu)