

Introduction to Custom Templates



Objectives

In this chapter you'll:

- Use class templates to create groups of related classes.
- Distinguish between class templates and class-template specializations.
- See how nontype template parameters can be used in place of constants declared inside a class.
- Learn about default template arguments.
- Learn about overloading function templates.













2 Chapter 18 Introduction to Custom Templates

Self-Review Exercises

- **18.1** State which of the following are *true* and which are *false*. If *false*, explain why.
 - a) Keywords typename and class as used with a template type parameter specifically mean "any user-defined class type."
 - **ANS:** False. Keywords typename and class in this context also allow for a type parameter of a fundamental type.
 - b) A function template can be overloaded by another function template with the same function name.

ANS: True.

- c) Template parameter names among template definitions must be unique.
- ANS: False. Template parameter names among function templates need not be unique.
- d) Each member-function definition outside its corresponding class template definition must begin with template and the same template parameters as its class template.

ANS: True.

18.2

Fill in the blanks in each of the following:
a) Templates enable us to specify, with a single code segment, an entire range of related
functions called, or an entire range of related classes called
ANS: function-template specializations, class-template specializations.

b) All template definitions begin with the keyword ______, followed by a list of template parameters enclosed in _____.

ANS: template, angle brackets (< and >).

The related functions generated from a function template all have the same name, so
the compiler uses ______ resolution to invoke the proper function.

ANS: overload.

d) Class templates also are called _____ types.

ANS: parameterized.

e) The ______ operator is used with a class-template name to tie each member-function definition to the class template's scope.

ANS: scope resolution.

Exercises

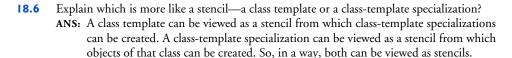
NOTE: Solutions to the programming exercises are located in the ch18solutions folder.

- **18.3** (Operator Overloads in Templates) Write a simple function template for predicate function isEqualTo that compares its two arguments of the same type with the equality operator (==) and returns true if they are equal and false otherwise. Use this function template in a program that calls isEqualTo only with a variety of fundamental types. Now write a separate version of the program that calls isEqualTo with a user-defined class type, but does not overload the equality operator. What happens when you attempt to run this program? Now overload the equality operator (with the operator function) operator==. Now what happens when you attempt to run this program?
 - ANS: If the user-defined class does not overload the equality operator (i.e., comment out lines 30–33 in the solution below), the compiler would report an error indicating that the class does not define this operator or a conversion to a type acceptable to the predefined operator. Once the class overloads the equality operator, the program should run fine.
- 18.5 Distinguish between the terms "function template" and "function-template specialization." ANS: A function template is used to instantiate function-template specializations.



Exercises





18.7 What's the relationship between function templates and overloading?

ANS: Function templates create function-template specializations—these are overloaded versions of a function. The main difference is at compile time, where the compiler automatically creates the code for the template functions from the function template rather than the programmer having to write the code.

18.8 The compiler performs a matching process to determine which function-template specialization to call when a function is invoked. Under what circumstances does an attempt to make a match result in a compile error?

ANS: The compiler tries to find a single best match using either an existing function or by creating a function template specialization. If there is no single best match, the compiler generates an error.

18.9 Why is it appropriate to refer to a class template as a parameterized type?

ANS: When creating specializations of a class template, it is necessary to provide a type (or possibly several types) to complete the definition of the new type being declared. For example, when creating an "array of integers" from an Array class template, the type int can be provided to the class template to complete the definition of an array of integers.

18.10 Explain why a C++ program would use the statement

Array<Employee> workerList{100};

ANS: When creating class-template specializations from a class template, it is necessary to provide a type (or possibly several types) to complete the definition of the new type being declared. For example, when creating an "array of Employees" from an Array class template, the type Employee is provided to the class template to complete the definition of an array of Employees. In this case, the number of Employees is known in advance.

18.11 Review your answer to Exercise ANS: . Explain why a C++ program might use the statement

Array<Employee> workerList;

ANS: Declares an Array object to store Employees. The default constructor is used. The number of Employees is not known in advance.

18.12 Explain the use of the following notation in a C++ program:

template<typename T> Array<T>::Array(int s)

ANS: This notation is used to begin the definition of the Array(int) constructor for the class template Array<T>.

18.13 Why might you use a nontype parameter with a class template for a container such as an array or stack?

ANS: To specify at compile time the size of the container class object being declared.





