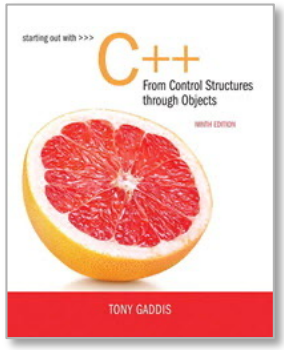


16.2

Function Templates

Function Templates

- Function template: a pattern for a function that can work with many data types
- When written, parameters are left for the data types
- When called, compiler generates code for specific data types in function call

Function Template Example

```
template <class T>

T times10(T num)
{
        return 10 * num;
}
```

template prefix

generic data type

type parameter

What gets generated when `times10` is called with an `int`:	What gets generated when `times10` is called with a `double`:

`int times10(int num)`
`{`
`  return 10 * num;`
`}`
	`double times10(double num)`
	`{`
	`  return 10 * num;`
	`}`

Function Template Example

```
template <class T>
T times10(T num)
{
        return 10 * num;
}
```

- Call a template function in the usual manner:

```
int ival = 3;
double dval = 2.55;
cout << times10(ival); // displays 30
cout << times10(dval); // displays 25.5
```

Function Template Notes

- Can define a template to use multiple data types:

```
template<class T1, class T2>
```

- Example:

```
template<class T1, class T2>      // T1 and T2 will be
double mpg(T1 miles, T2 gallons) // replaced in the
{                                 // called function
return miles / gallons          // with the data
}                                 // types of the
                                  // arguments
```

Function Template Notes

- Function templates can be overloaded Each template must have a unique parameter list

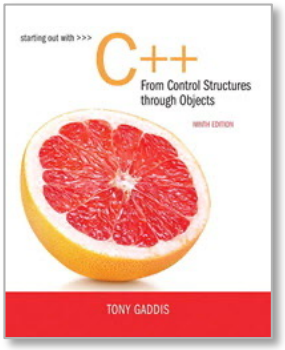
```
template <class T>
T sumAll(T num) ...
template <class T1, class T2>
T1 sumall(T1 num1, T2 num2) ...
```

Function Template Notes

- All data types specified in template prefix must be used in template definition
- Function calls must pass parameters for all data types specified in the template prefix
- Like regular functions, function templates must be defined before being called

Function Template Notes

- A function template is a pattern
- No actual code is generated until the function named in the template is called
- A function template uses no memory
- When passing a class object to a function template, ensure that all operators in the template are defined or overloaded in the class definition

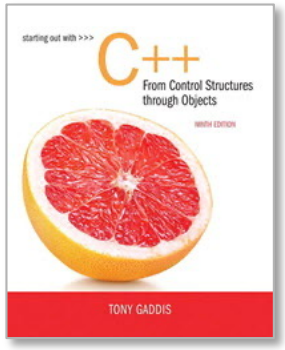


16.3

Where to Start When Defining Templates

Where to Start When Defining Templates

- Templates are often appropriate for multiple functions that perform the same task with different parameter data types
- Develop function using usual data types first, then convert to a template:
  - add template prefix
  - convert data type names in the function to a type parameter (*i.e.*, a T type) in the template



16.4

Class Templates

Class Templates

- Classes can also be represented by templates. When a class object is created, type information is supplied to define the type of data members of the class.
- Unlike functions, classes are instantiated by supplying the type name (`int`, `double`, `string`, etc.) at object definition

Class Template Example

```
template <class T>
class grade
{
    private:
        T score;
    public:
        grade(T);
        void setGrade(T);
        T getGrade()
};
```

Class Template Example

- Pass type information to class template when defining objects:

```
grade<int> testList[20];

grade<double> quizList[20];
```

- Use as ordinary objects once defined

Class Templates and Inheritance

- Class templates can inherit from other class templates:

```
template <class T>
class Rectangle
  { ... };
template <class T>
class Square : public Rectangle<T>
  { ... };
```

- Must use type parameter `T` everywhere base class name is used in derived class