**Chapter 23: Introduction to Custom Templates**

**Section 23.1 Introduction**

23.1 Q1: The relationship between function templates and function-template specializations is most similar to the relationship between:

1. Classes and objects.
2. Classes and functions.
3. Functions and return types.
4. Headers and source files.

**ANS a. Classes and objects.**

**Section 23.2 Class Templates**

23.2 Q1: Class templates:

1. May include the statement template< typename Type >anywhere.
2. Must put template< typename Type >before the class definition.
3. Must include template< typename Type >inside the class definition.
4. Have the option of including the optional statement template< typename Type >.

**ANS: b. Must put template< typename Type > before the class definition.**

23.2 Q2: For a class template, the scope resolution operator (::) is needed:

1. Only in the definitions of the member functions defined outside the class.
2. *Both* in the prototype and definition of a member function.
3. Only if *multiple* class-template specializations will be created from this class template.
4. In *neither* the definition *nor* prototype of member functions.

**ANS: a. Only in the definitions of the member functions defined outside the class.**

23.2 Q3: Function templates:

1. Can include objects of template classes as parameters.
2. Must have return type T.
3. Do *not* need a separate template< typename type >statement if they take objects from a template class as a parameter.
4. Do *not* need a separate template< typename type >statement.

**ANS: a. Can include objects of template classes as parameters.**

**Section 23.3 Function Template to Manipulate a Class-Template Specialization Object**

**No questions.**

**Section 23.4 Nontype Parameters**

23.4 Q1: Nontype parameters are:

1. Unable to have default arguments.
2. Specified before the angle-bracket-enclosed type-parameter list.
3. Constants.
4. Required for class templates.

**ANS: c. Constants.**

**Section 23.5 Default Arguments for Template Type Parameters**

23.5 Q1: Which of the following is *true*?

1. Only class templates may specify default type arguments for type parameters.
2. Only function templates may specify default type arguments for type parameters.
3. Both class templates and function templates may specify default type arguments for type parameters.
4. None of the above.

**ANS: c. Both class templates and function templates may specify default type arguments for type parameters..**

23.5 Q2: Default type parameters are allowed *only*:

1. If the class template also has nontype parameters.
2. If the class template does *not* have any nontype parameters.
3. If the class is used as a container class.
4. As the rightmost (trailing) parameters in a template’s type-parameter list.

**ANS: d. As the rightmost (trailing) parameters in a template’s type-parameter list.**

**Section 23.6 Overloading Function Templates**

23.6 Q1: A function template can be overloaded by:

1. Using other function templates with the same function name and parameters.
2. Using non-template functions with the same name and different parameters.
3. Using non-template functions with a different name but the same parameters.
4. Using other function templates with a different name but the same parameters.

**ANS b. Using non-template functions with the same name and different parameters.**

23.6 Q2: Assuming that all four of the following functions are defined, which one will be called by the function call square( 23.4 )?

1. template< typename T >  
   T square( T num )
2. template< typename T1, typename T2 >  
   T1 square( T1 num1, T2 num2 )
3. int square( int num )
4. double square( double num )

**ANS d. double square( double num )**