**Chapter 22: Exception Handling: A Deeper Look**

**Section 22.1 Introduction**

22.1 Q1: Exception handling may allow a program to:

1. Terminate in a controlled manner.
2. Be more robust and fault-tolerant.
3. Continue executing as if no problem was encountered.
4. All of the above.

**ANS d. All of the above.**

**Section 22.2 Example: Handling an Attempt to Divide by Zero**

22.2 Q1: The correct order in which an exception is detected and handled is:

1. try, catch, throw
2. throw, catch, try
3. catch, throw, try
4. try, throw, catch

**ANS d. try, throw, catch**

22.2 Q2: Once an exception is thrown, when can control *return* to the throw point?

1. Never.
2. Only after the exception is caught.
3. Once the stack unwinding process is completed.
4. Immediately after the exception is thrown.

**ANS a. Never.**

22.2 Q3: The try block *cannot*:

1. Enclose the code that may throw the exception.
2. Enclose its own catch blocks.
3. Test enclosing try blocks for additional catch statements if this try block’s catch statements can’t match the exception being thrown.
4. Have exceptions explicitly or implicitly thrown in the try block itself.

**ANS b. Enclose its own catch blocks.**

22.2 Q4: catch blocks are *not* required to contain:

1. Braces { }.
2. Parentheses ( ).
3. Some form of parameter type indication.
4. A parameter name.

**ANS d. A parameter name.**

22.2 Q5: An exception:

1. Terminates program execution.
2. Terminates the block where the exception occurred.
3. Will terminate the block where the exception occurred unless a catch command stops it.
4. Will *not* terminate a block unless *explicitly* instructed to do so.

**ANS b. Terminates the block where the exception occurred.**

**Section 22.3 Rethrowing an Exception**

22.3 Q1: To *rethrow* an exception, the exception handler must:

1. Use the throw; statement.
2. Use the throw command with the same parameters as the original exception.
3. Return a reference to whatever caused the original exception.
4. Not have attempted to process that exception at all.

**ANS a. Use the throw; statement.**

22.3 Q2: Select the *false* statement. A rethrown exception:

1. Is detected by the *next* enclosing try block.
2. Is the immediate result of a throwcommand.
3. Can be processed by exception handlers following the enclosing try block.
4. Must have been fully processed at the time it was rethrown.

**ANS d. Must have been fully processed at the time it was rethrown.**

**Section 22.4 Stack Unwinding**

22.4 Q1: The purpose of stack unwinding is to:

1. Attempt to catch exceptions that are not caught in their scope.
2. Improve catchblocks by allowing them to handle multiple exceptions.
3. Return control to the function that created the exception.
4. Aid the terminate command in shutting down the program.

**ANS: a. Attempt to catch exceptions that are not caught in their scope.**

**Section 22.5 When to Use Exception Handling**

22.5 Q1: Exception handling should *not* be used:

1. As an alternative for program control.
2. To make error handling uniform on large projects.
3. To deal with errors that do *not* arise very often.
4. To deal with errors for components that will be widely used in other applications, such as classes and libraries.

**ANS a. As an alternative for program control.**

**Section 22.6 Constructors, Destructors and Exception Handling**

22.6 Q1: Select the *false* statement. If an exception is thrown from a constructor:

1. The object being constructed will not be constructed.
2. For an array, destructors for all array elements are called, even if those array elements have not yet been constructed.
3. The exception can contain the error information that the constructor would not be able to return in the normal manner.
4. For an object with member objects, and whose outer object has not been constructed, the destructor is called for the member objects.

**ANS: b. For an array, destructors for all array elements are called, even if those array elements have not yet been constructed.**

**Section 22.7 Exceptions and Inheritance**

22.7 Q1: An advantage of using inheritance with exceptions is:

1. The ability to catchrelated errors easily.
2. Allowing catch statements to be imported into classes.
3. The ability to explicitly test for derived class objects individually.
4. The simplification of destructor calls for objects.

**ANS: a. The ability to catch related errors easily.**

**Section 22.8 Processing new Failures**

22.8 Q1: Select the *false* statement. Depending on the compiler:

1. A failed new operation can return a 0.
2. A failed new operation can throw a bad\_alloc exception.
3. A failed new operation can throw an exception if the <new> header file has been included.
4. A failed new operation can automatically be caught at compile time.

**ANS: d. A failed new operation can automatically be caught at compile time.**

22.8 Q2: Select the *false* statement. The new operator:

1. Can attempt to allocate as much memory as the programmer requests.
2. Returns a pointer to a location in memory.
3. Can indicate failure differently on different compilers.
4. Throws a bad\_alloc exception regardless of what function is registered with set\_new\_handler.

**ANS: d. Throws a bad\_alloc exception regardless of what function is registered with set\_new\_handler.**

**Section 22.9 Class unique\_ptr and Dynamic Memory Allocation**

22.9 Q1: If dynamic memory has been allocated for an object and an exception occurs, then:

1. The catch block will not work properly.
2. A memory leak could result.
3. The object's constructor will cause another exception.
4. Multiple pointers to memory could be created.

**ANS: b. A memory leak could result.**

22.9 Q2: Which statement about class unique\_ptr (of the new C++ standard) and dynamic memory allocation is *false*?

a. An object of class unique\_ptr maintains a pointer to dynamically allocated memory.

b. When a unique\_ptr object destructor is called (for example, when a unique\_ptr object goes out of scope), it performs a destroy operation on its pointer data member.

c. Class template unique\_ptr provides overloaded operators \* and -> so that a unique\_ptr object can be used just as a regular pointer variable is.

d. Class unique\_ptr is part of the new C++ standard and it replaces the deprecated auto\_ptr class.

**ANS: b. When a unique\_ptr object destructor is called (for example, when a unique\_ptr object goes out of scope), it performs a destroy operation on its pointer data member. [The operation is delete.]**

**Section 22.10 Standard Library Exception Hierarchy**

22.10 Q1: Select the *false* statement regarding exceptions.

1. The C++ standard has a hierarchy of exception classes.
2. All exception classes are accessible via <exception>.
3. Several classes derive from class exception.
4. The what function can be overridden in each class derived from exception.

**ANS: b. All exception classes are accessible via <exception>.**

22.10 Q2: Which class indicates that an error occurred in which an arithmetic result was larger than the largest number that can be stored in the computer?

1. invalid\_argument.
2. bad\_exception.
3. out\_of\_range.
4. overflow\_error.

**ANS: d. overflow\_error.**