Section 16.1 Introduction

16.1 Q1: Exception handling may allow a program to:

All of the above.

Be more robust and fault-tolerant.

Continue executing as if no problem was encountered.

Terminate in a controlled manner.

End of Question 1

All of the above.

Question 2.

Section 16.2 Exception-Handling Overview

16.2 Q1: Which of the following is not one of the disadvantages of not using exception-handling to deal with errors?

Frequent tests for infrequently occurring errors can degrade a program’s performance.

Intermixing program logic with error-handling logic can make the program more difficult to read, modify, maintain and debug.

All of the above are disadvantages of not using exception-handling to deal with errors.

Programmers may delay writing error-processing code or sometimes forget to include it.

End of Question 2

All of the above are disadvantages of not using exception-handling to deal with errors.

Question 3.

Section 16.3 Example: Handling an Attempt to Divide by Zero

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

try, throw, catch.

try, catch, throw.

throw, catch, try.

catch, throw, try.

End of Question 3

try, throw, catch

Question 4.

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

Once the stack unwinding process is completed.

Never.

Only after the exception is caught.

Immediately after the exception is thrown.

End of Question 4  
Only after the exception is caught.

Question 5.

16.3 Q3: The try block cannot:

Enclose the code that may throw the exception.

Test enclosing try blocks for additional catch statements if this try block’s catch statements can’t match the exception being thrown.

Have exceptions explicitly or implicitly thrown in the try block itself.

Enclose its own catch blocks.

End of Question 5

Enclose its own catch blocks.

Question 6.

16.3 Q4: catch blocks are not required to contain:

Some form of parameter type indication.

Braces { }.

A parameter name.

Parentheses ( ).

End of Question 6

A parameter name.

Question 7.

16.3 Q5: An exception:

Will terminate the block where the exception occurred unless a catch command stops it.

Terminates the block where the exception occurred.

Terminates program execution.

Will not terminate a block unless explicitly instructed to do so.

End of Question 7

Terminates the block where the exception occurred.

Question 8.

Section 16.4 When to Use Exception Handling

16.4 Q1: Exception handling should not be used:

As an alternative for program control.

To deal with errors for components that will be widely used in other applications, such as classes and libraries.

To deal with errors that do not arise very often.

To make error handling uniform on large projects.

End of Question 8

As an alternative for program control.

Question 9.

Section 16.5 Rethrowing an Exception

16.5 Q1: To rethrow an exception, the exception handler must:

Use the throw command with the same parameters as the original exception.

Return a reference to whatever caused the original exception.

d. Not have attempted to process that exception at all.

Use the throw; statement.

End of Question 9

Use the throw; statement.

Question 10.

16.5 Q2: Select the false statement. A rethrown exception:

Is detected by the next enclosing try block.

Can be processed by exception handlers following the enclosing try block.

Is the immediate result of a throw; command.

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

End of Question 10

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

Question 11.

Section 16.6 Exception Specifications

16.6 Q1: The proper syntax for a throw list is:

int g( double h )

throw ( a )

throw ( b )

throw ( c ).

a.int g( double h )

throw ( a, b, c ).

int g( double h )

throw ( a ),

throw ( b ),

throw ( c ).

int g( double h )

throw ( a, b, c ).

End of Question 11

int g( double h )  
throw ( a, b, c ).

Question 12.

16.6 Q2: Placing throw() after a function's parameter list:

Guarantees that only programmer-defined exceptions can be thrown in this function.

d. Indicates that the compiler will issue an error if the function contains a throw expression.

Guarantees that all exceptions can be thrown in this function.

Indicates that throwing an exception in this function would call unexpected.

End of Question 12

Indicates that throwing an exception in this function would call unexpected.

Question 13.

Section 16.7 Processing Unexpected Exceptions

16.7 Q1: Select the false statement. The functions set\_terminate and set\_unexpected:

Take as arguments pointers to void functions with no arguments.

Return pointers to the last function called by terminate and unexpected, respectively.

d. Have their prototypes in header file .

Each return 0 the first time they are called.

End of Question 13

d. Have their prototypes in header file .

Question 14.

16.7 Q2: Which of the following is not a case in which function terminate is called?

When an attempt is made to rethrow an exception when there is no exception currently being handled.

When the exception mechanism cannot find a matching catch for a thrown exception.

When a destructor attempts to throw an exception during stack unwinding.

When the abort function is called before any call to function set\_abort.

End of Question 14

When the abort function is called before any call to function set\_abort.

Question 15.

Section 16.8 Stack Unwinding

16.8 Q1: The purpose of stack unwinding is to:

Return control to the function that created the exception.

b. Improve catch blocks by allowing them to handle multiple exceptions.

Attempt to catch exceptions that are not caught in their scope.

Aid the terminate command in shutting down the program.

End of Question 15

Attempt to catch exceptions that are not caught in their scope.

Question 16.

Section 16.9 Constructors, Destructors and Exception Handling

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

The exception can contain the error information that the constructor would not be able to return in the normal manner.

For an object with member objects, and whose outer object has not been constructed, the destructor is called for the member objects.

The object being constructed will not be constructed.

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

End of Question 16

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

Question 17.

Section 16.10 Exceptions and Inheritance

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

The simplification of destructor calls for objects.

Allowing catch statements to be imported into classes.

The ability to catch related errors easily.

The ability to explicitly test for derived class objects individually.

End of Question 17

The ability to catch related errors easily.

Question 18.

Section 16.11 Processing new Failures

16.11 Q1: Select the false statement. Depending on the compiler:

A failed new command can automatically be caught at compile time.

A failed new command can throw a bad\_alloc exception.

A failed new command can throw an exception if the header file has been included.

A failed new command can return a 0.

End of Question 18

A failed new command can automatically be caught at compile time.

Question 19.

16.11 Q2: Select the false statement. The new operator:

Can indicate failure differently on different compilers.

Returns a pointer to a location in memory.

Can attempt to allocate as much memory as the programmer requests.

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

End of Question 19

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

Question 20.

Section 16.12 Class auto\_ptr and Dynamic Memory Allocation

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

The object's constructor will cause another exception.

A memory leak could result.

Multiple pointers to memory could be created.

The catch block will not work properly.

End of Question 20

A memory leak could result.

Question 21.

Section 16.13 Standard Library Exception Hierarchy

16.13 Q1: Select the false statement regarding exceptions.

The C++ standard has a hierarchy of exception classes.

Several classes derive from class exception.

All exception classes are accessible via .

The what function can be overridden in each class derived from exception.

End of Question 21

All exception classes are accessible via .

Question 22.

16.13 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?

invalid\_argument.

out\_of\_range.

bad\_exception.

overflow\_error.

End of Question 22

overflow\_error.

Question 23.

Section 16.14 Other Error-Handling Techniques

16.14 Q1: Which of the following is not an error-handling technique?

set\_new\_handler.

longjump.

exit.

ifndef.

End of Question 23

ifndef.

Question 24.

16.14 Q2: Both “ignoring the exception” and “aborting the program” are error-handling techniques that:

Allow program execution to proceed as if no error had occurred.

Always result in a resource leak.

Should not be used for mission-critical applications.

Cannot be used if the error is fatal.

Should not be used for mission-critical applications.