**Chapter 17: Classes: A Deeper Look; Throwing Exceptions**

**Section 17.2 Time Class Case Study**

17.2 Q1: Member access specifiers (public and private) can appear:

1. In any order and multiple times.
2. In any order (public first or private first) but not multiple times.
3. In any order and multiple times, if they have brackets separating each type.
4. Outside a class definition.

**ANS: a. In any order and multiple times.**

17.2 Q2: Which of the following preprocessor directives does *not* constitute part of the preprocessor wrapper?

1. #define
2. #endif
3. #ifndef
4. #include

**ANS: d. #include**

17.2 Q3: Member function definitions:

1. Always require the scope resolution operator (::).
2. Require the scope resolution operator only when being defined outside of the definition of their class.
3. Can use the scope resolution operator anywhere, but become public functions.
4. Must use the scope resolution operator in their function prototype.

**ANS: b. Require the scope resolution operator only when being defined outside of the definition of their class.**

17.2 Q4: Parameterized stream manipulator setfill specifies the fill character that’s displayed when an output is displayed in a field wider than the number of characters or digits in the output. The effect of setfill applies:

1. Only to the current value being displayed.
2. Only to outputs displayed in the current statement.
3. Until explicitly set to a different setting.
4. Until the output buffer is flushed.

**ANS: c. Until explicitly set to a different setting.**

17.2 Q5: Every object of the same class:

1. Gets a copy of every member function and member variable.
2. Gets a copy of every member variable.
3. Gets a copy of every member function.
4. Shares pointers to all member variables and member functions.

**ANS: b. Gets a copy of every member variable.**

17.2 Q6: A class’s functions can throw exceptions, such as \_\_\_\_\_\_\_\_\_\_to indicate invalid data.

a. invalid\_data

b. bad\_data

c. invalid\_argument

d. bad\_argument

ANS: c. invalid\_argument

**Section 17.3 Class Scope and Accessing Class Members**

17.3 Q1: Variables defined inside a member function of a class have:

1. File scope.
2. Class scope.
3. Block scope.
4. Class or block scope, depending on whether the binary scope resolution operator (::) is used.

**ANS: c. Block scope.**

17.3 Q2: A class-scope variable hidden by a block-scope variable can be accessed by preceding the variable name with the class name followed by:

1. ::
2. :
3. .
4. ->

**ANS: a. ::**

**Section 17.4 Access Functions and Utility Functions**

17.4 Q1: The type of function a client would use to check the balance of a bank account would be:

1. A utility function.
2. A predicate function.
3. An access function.
4. A constructor.

**ANS: c. an access function.**

17.4 Q2: Utility functions:

1. Are private member functions that support operations of the class’s other member functions.
2. Are part of a class’s interface.
3. Are intended to be used by clients of a class.
4. Are a type of constructor.

**ANS: a. Are private** **member functions that support operations of the class’s other ­­member functions.**

**Section 17.5 Time Class Case Study: Constructors with Default Arguments**

17.5 Q1: A default constructor:

1. Is a constructor that must receive no arguments.
2. Is the constructor generated by the compiler when no constructor is provided by the programmer.
3. Does not perform any initialization.
4. Both (a) and (b).

**ANS: d. Both (a) and (b).**

17.5 Q2: If a member function of a class already provides all or part of the functionality required by a constructor or another member function then:

1. Copy and paste that member function’s code into this constructor or member function.
2. Call that member function from this constructor or member function.
3. That member function is unnecessary.
4. This constructor or member function is unnecessary.

**ANS: b. Call that member function from this constructor or member function.**

17.5 Q3[C++11]: Assuming the following constructor is provided for class Time

explicit Time( int = 0, int = 0, int = 0 );

which of the following is *not* a valid way to initialize a Time object?

1. Time t1;
2. Time t2{ 22, 40 };
3. Time t3( 22, 40 );
4. (a), (b) and (c) are all valid ways to initialize a Time object.

**ANS: d. (a), (b) and (c) are all valid ways to initialize a Time object.**

17.5 Q3[C++11]: Which of the following statements is *false*?

1. You can overload a classes constructors.
2. There is no mechanism in C++ for a constructor to call another constructor in the same class.
3. Just as a constructor can call a class’s other member functions to perform tasks, C++11 allows constructors to call other constructors in the same class.
4. To overload a constructor, provide in the class definition a prototype for each version of the constructor, and provide a separate constructor definition for each overloaded version.

**ANS: b. There is no mechanism in C++ for a constructor to call another constructor in the same class. [C++ includes delegating constructors—that is, constructors that can delegate their work to other constructors in the same class.]**

**Section 17.6 Destructors**

17.6 Q1: Which of the following is *not* true of a constructor and destructor of the same class?

1. They both have the same name aside from the tilde (~) character.
2. They are both usually called once per object created.
3. They both are able to have default arguments.
4. Both are called automatically, even if they are not explicitly defined in the class.

**ANS: c. They both are able to have default arguments.**

17.6 Q2: Which of the following is *not* true of a destructor?

1. It performs termination housekeeping.
2. It is called before the system reclaims the object’s memory.
3. If the programmer does not explicitly provide a destructor, the compiler creates an “empty” destructor.
4. It releases the object’s memory.

**ANS: d. It releases the object’s memory.**

**Section 17.7 When Constructors and Destructors Are Called**

17.7 Q1: Given the class definition:

class CreateDestroy

{

public:

CreateDestroy() { cout << "constructor called, "; }

~CreateDestroy() { cout << "destructor called, "; }

};

What will the following program output?

int main()

{

CreateDestroy c1;

CreateDestroy c2;

return 0;

}

1. constructor called, destructor called, constructor called, destructor called,
2. constructor called, destructor called,
3. constructor called, constructor called,
4. constructor called, constructor called, destructor called, destructor called,

**ANS: d. constructor called, constructor called, destructor called, destructor called,**

17.7 Q2: Given the class definition:

class CreateDestroy

{

public:

CreateDestroy() { cout << "constructor called, "; }

~CreateDestroy() { cout << "destructor called, "; }

};

What will the following program output?

int main()

{

for ( int i = 1; i <= 2; ++i )

CreateDestroy cd;

return 0;

}

1. constructor called, destructor called, constructor called, destructor called,
2. constructor called, constructor called,
3. constructor called, constructor called, destructor called, destructor called,
4. Nothing.

**ANS: a. constructor called, destructor called, constructor called, destructor called,**

**Section 17.8 Time Class Case Study: A Subtle Trap—Returning a Reference to a private Data Member**

17.8 Q1: Returning references to non-const, private data:

1. Allows private functions to be modified.
2. Is only dangerous if the binary scope resolution operator (::) is used in the function prototype.
3. Allows private member variables to be modified, thus “breaking encapsulation.”
4. Results in a compiler error.

**ANS: c. Allows private member variables to be modified, thus “breaking encapsulation.”**

17.8 Q2: A client changing the values of private data members is:

1. Only possible by calling private member functions.
2. Possible using public functions and references.
3. Never possible.
4. Only possible if the private variables are not declared inside the class.

**ANS: b. Possible using public functions and references.**

**Section 17.9 Default Memberwise Assignment**

17.9 Q1: The assignment operator (=) *can* be used to:

1. Test for equality.
2. Copy data from one object to another.
3. Compare two objects.
4. Copy a class.

**ANS: b. Copy data from one object to another.**

**Section 17.10 const (Constant) Objects and const Member Functions**

17.10 Q1: Which of the following statements will *not* produce a syntax error?

1. Defining a const member function that modifies a data member of the object.
2. Invoking a non-const member function on a const object.
3. Declaring an object to be const.
4. Declaring a constructor to be const.

**ANS c. Declaring an object to be const.**

**17.10 Q2: The code fragment:**

Increment::Increment( int c, int i )

: increment ( i )

{

count = c;

}

does *not* cause any compilation errors. This tells you that:

1. count must be a non-const variable.
2. count must be a const variable.
3. increment must be a non-const variable.
4. increment must be a const variable.

**ANS a. count must be a non-const variable.**

**Section 17.11 Composition: Objects as Members of Classes**

17.11 Q1: When composition (one object having another object as a member) is used:

1. The host object is constructed first and then the member objects are placed into it.
2. Member objects are constructed first, in the order they appear in the host constructor’s initializer list.
3. Member objects are constructed first, in the order they are declared in the host’s class.
4. Member objects are destructed last, in the order they are declared in the host’s class.

**ANS c. Member objects are constructed first, in the order they are declared in the host’s class.**

17.11 Q2: An error occurs if:

1. A non-reference, non-const, primitive data member is initialized in the member initialization list.
2. An object data member is not initialized in the member initialization list.
3. An object data member does not have a default constructor.
4. An object data member is not initialized in the member initialization list and does not have a default constructor.

**ANS d. An object data member is not initialized in the member initialization list and does not have a default constructor.**

**Section 17.12 friend Functions and friend Classes**

17.12 Q1: If the line:

friend class A;

appears in class B, and the line:

friend class B;

appears in class C, then:

1. Class A is a friend of class C.
2. Class A can access private variables of class B.
3. Class C can call class A’s private member functions.
4. Class B can access class A’s private variables.

**ANS: b. Class A can access private variables of class B.**

17.12 Q2: Which of the following statements about friend functions and friend classes is false?

1. A class can either grant friendship to or take friendship from another class using the friend keyword.
2. A friend declaration can appear *anywhere* in a class definition.
3. A friend of a class can access all of its private data member and member functions.
4. The friendship relationship is neither symmetric nor transitive.

**ANS a. A class can either grant friendship to or take friendship from another class using the friend keyword.**

**Section 17.13 Using the this Pointer**

17.13 Q1: For a *non-constant* member function of class Test, the this pointer has type:

1. const Test \*
2. Test \* const
3. Test const \*
4. const Test \* const

**ANS: b. Test \* const**

17.13 Q2: Inside a function definition for a member function of an object with data member x, which of the following is *not* equivalent to this->x:

1. \*this.x
2. (\*this).x
3. x
4. None of the above are equivalent.

**ANS: a. \*this.x**

17.13 Q3: Assume that t is an object of class Test, which has member functions a(), b(), c() and d(). If the functions a(), b() and c() all return references to an object of class Test (using the dereferenced this pointer) and function d() returns void, which of the following statements will *not* produce a syntax error:

1. t.a().b().d();
2. a().b().t;
3. t.d().c();
4. t.a().t.d();

**ANS: a. t.a().b().d();**

**Section 17.14 static Class Members**

17.14 Q1: If Americans are objects of the same class, which of the following attributes would most likely be represented by a staticvariable of that class?

1. Age.
2. The President.
3. Place of birth.
4. Favorite food.

**ANS: b. The President.**

17.14 Q2: static data members of a certain class:

1. Can be accessed only if an object of that class exists.
2. Cannot be changed, even by objects of the same *that* class.
3. Have class scope.
4. Can only be changed by static member functions.

**ANS: c. Have class scope.**

17.14 Q3: static member functions:

1. Can use the this pointer.
2. Can access only other static member functions and static data members.
3. Cannot be called until an object of their class is instantiated.
4. Can be declared const as well.

**ANS: b. Can only access other static member functions and static data members.**