Q: Classes **cannot**

1. Be derived from(來自) other classes
2. Initialize(初始化) data members in the class definition
3. Be used to model attributes and behaviors of object
4. Include objects from other classes as member

Ans: b.

Q: A default constructor has how many parameters(參數)?

1. **0**
2. 1
3. 2
4. Variable number

Ans: a.

Q: A constructor can specify(**指定**) the return type:

1. Int.
2. String.
3. Void.
4. **A constructor cannot specify a return type.**

Ans: d.

Q: The compiler will implicitly(**暗中地**) create a default constructor if:

1. The class does not contain any data members
2. The programmer specifically requests that the compiler do so
3. The class **does not** define any constructors
4. The class already defines a default constructor

And: c.

Q: 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.

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

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

Ans: a.

Q: In the **source-code(.cpp)** containing a class’s member function definitions, each member function definition must be tied to the class definition by preceding the member function name with the class name and **::** , which is known as the:

1. Member definition linker
2. Class implementation connector
3. Source code resolver
4. **Binary scope resolution operator**

Ans: d.

Q: When a client code programmer uses a class whose **implementation(實行)** is in a separate file from its interface, that implementation code is **merged with(合併)** the client’s code during the:

1. Programming phase
2. Compiling phase
3. **Linking phase**
4. Executing phase

Ans: c.

Q: When independent software vendors(供應商) provide class libraries to clients, they typically(通常) give the \_\_\_\_\_\_\_\_\_ interface and the \_\_\_\_\_\_\_\_ for the class’s implementation.

1. Source code file, Source code file.
2. **Source code file, Object file.**
3. Object file, Source code file.
4. Object file, Object file.

Ans: b.

Q: Which of the following statements about **separating** a class’s interface and implementation is **not** true?

1. **Inline member function definitions are included in the header file**
2. **Changes in the class’s interface(.h檔) will affect the client**
3. **Private data members are included in the header file**
4. Changes in the class’s implementation will affect the client

Ans: d

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

1. A utility function // private function, 提供 public 使用
2. A predicate function //判斷true or false
3. **An access function // 存取**
4. A constructor // 建構物件

Ans: c.

Q: 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.

Q: Which of the following statements is not true of a constructor 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. The both are able to have default arguments
4. Both are called automatically ,even if they are not explicitly defined in the class

Ans: c.

Q: Which of the following statements is not true of a destructor?

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

Ans: d.

Q: 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,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.

Q: What type of member functions allow a client of a class to assign values to private data member?

1. Client member functions
2. Get member functions
3. Set member functions
4. None of the above

Ans: c.

Q: Which of the following statements is false?

1. Always try to localize the effects of changes to a class’s data members by accessing and manipulating the data members through their set and get functions
2. Services useful to the client should typically be provided in the class’s private interface. Change is the rule rather than the exception. You should anticipate that your code will be modified.
3. Providing public set and get functions does allow clients of the class to access hidden data, but only indirectly.

Ans: b.

Q: A client changing the values of private data members is:

1. Only possible by calling private member function
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.

Q: 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’ member functions

Ans: b.

Q: 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.

Increment :: Increment(int c,int i) : Increment(i){

count = c;

}

Dose 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.

Q: What 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 initialized
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.

Q: 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.

Q: 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.

Q: Which of the following is not true about friend functions and friend classes?

1. A class can either grant friendship to or take friendship form another class using …
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 function
4. The friendship relationship is neither symmetric nor transitive

Ans: a.

Int\* - pointer to int

Int const\* - pointer to const int

Int\* const – const pointer to int

Int const\* const – const pointer to cons tint

Now the first const can be on either side of the type so:

Const int\* == int const\*

Const int\* const == int const\* const

Const TYPE value;

TYPE const value;

兩種寫法一樣。含義是：const修飾的類型為TYPE的變數value是不可變的。

對於一個非指標的類型TYPE，無論怎麼寫，都是value不可變。

Const int nValue; // nValue is const

Int const nValue; // nValue is const

對於指標類型的TYPE，不同的寫法有不同的情形，例如：

1. Const char \*pContent;
2. Char \*const pContent;
3. Char const \*pContent;
4. Const char\* const pContent;

前三種寫法，可以把加上括弧

1. Const (char) \*pContent;
2. (char\*) const pContent;
3. (char) const \*pContent;

A,C const修飾的類型為char的變數\*pContent為常數，pContent的內容是常數不可變。

B還有一種寫法： const (char\*) pContent; 含義：const修飾的類型為 char … pContent 指標本身為常數不可變。

D 是 A和B的混合體，表示指標本身和指標內容兩者皆為常數不可變

* 1. 指標本身是常數不可變

(char\*) const pContent;

Const (char\*) pContent;

* 1. 指標所指向的內容是常數不可變

Const (char) \*pContent;

(char) const \*pContent;

* 1. 兩者都不可變

Const char\* const pContent;

Const 修飾函數參數

Const 修飾函數參數，表示函數中不能修改數值(包括參數本身的值或參數中包含的值)

Void function(const int var); //傳過來的參數在函數內不可改變(無意義，因var本身就是形參)

Void function(const char\* var); //參數指標所指內容為常數不可變

Void function(char\* const var); //參數指標本身為常數不可變(也無意義，因為char\* var 也是形參)

Q: 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.

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

1. \*this.x
2. (\*this).x
3. X
4. (\*(&(\*this))).x

Ans: a.

Q: Assume that t is an object of class Test. Which has member function a ( ), b ( ), c ( ) and d ( ). If the functions a( ), b( ) and c( ) 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.

Q: If Americans are objects of the same class, which of the following attributes would most likely represented by a static variable of that class?

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

Ans: b.

Q: 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.

Q: static member functions:

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

Ans: b.

Q: Which of the following operators cannot be overload?

1. The . operator
2. The -> operator
3. The & operator
4. The [ ] operator

Ans: a.

Q: Which statement about operator overloading is false?

1. New operators can never be created
2. Certain overloaded operators can change the number of arguments they take
3. The precedence of an operator cannot be changed by overloading
4. Overloading cannot change how an operator works on built-in type

Ans: b.

Q: An overloaded + operator takes a class object and a double as operands. For it to be commutative (i.e. a+b and b+a both work):

1. Operator + must be a member function of the class from which the objects are instantiated
2. Operator + must be a member function
3. It must be overloaded twice; the operator + function that takes the object as the left operand must be a member function, and the other operator + function must be a global function
4. The + operator cannot be overloaded to be commutative

Ans: c.

Q: Suppose you have a programmer – defined data type Data and want to overload the … to the screen in the form cout<< dataToPrint; and allow cascaded function calls … definition would be:

1. Ostream &operator << (ostream &output, const Data &dataToPrint);