1. Los métodos definidos en una clase derivada nunca pueden acceder a las propiedades privadas de una clase base.
2. Una clase abstracta se caracteriza por no tener definido ningún constructor.
3. La siguiente clase: `class S {public: S(); S(const S &s); virtual ~S();};` constituye una interfaz en C++.
4. Desde un método de una clase derivada nunca puede invocarse un método implementado con idéntica signatura de una de sus clases base.
5. Los métodos virtuales son métodos abstractos.
6. Los métodos abstractos son métodos con enlace dinámico.
7. Los constructores de las clases abstractas son métodos con enlace dinámico.
8. Un atributo de clase debe tener visibilidad pública para poder ser accedido por los objetos de la clase.
9. Un método sobrecargado es aquel que recibe como argumento al menos una variable polimórfica.
10. Un método tiene polimorfismo puro cuando devuelve una variable polimórfica.
11. En C++ no podemos hacer sobrecarga de operadores para tipos predefinidos.
12. En C++, si no se define un constructor sin argumentos explícitamente, el compilador proporciona uno por defecto.
13. En la misma clase, podemos definir constructores con distinta visibilidad.
14. Si no se captura una excepción lanzada por un método, el programa no advierte que ha ocurrido algún error y continua su ejecución normalmente.
15. La instrucción `throw` permite lanzar como excepción cualquier tipo de dato.
16. Las funciones genéricas no se pueden sobrecargar.
17. Dada una clase genérica, se pueden derivar de ella clases no genéricas.
18. De una clase abstracta no se pueden crear instancias, excepto si se declara explícitamente algún constructor.
19. Una clase interfaz no puede tener atributos de instancia. Una clase abstracta sí puede tenerlos.
20. La herencia de interfaz se implementa mediante herencia pública.

VERDADERO
FALSO
FALSO
FALSO
FALSO
VERDADERO

FALSO
FALSO
FALSO
FALSO
VERDADERO
FALSO

VERDADERO
FALSO
FALSO
VERDADERO
FALSO
VERDADERO

FALSO
VERDADERO
VERDADERO

1. La herencia pública permite a los métodos definidos en una clase derivada acceder a las propiedades privadas de la clase base.
2. Una clase abstracta se caracteriza por declarar al menos un método abstracto.
3. La siguiente clase: `class S {public: Object *o;};` constituye una clase interfaz en C++.
4. Los métodos abstractos siempre tienen enlace dinámico.
5. Los constructores siempre son métodos virtuales.
6. Un método tiene polimorfismo puro cuando tiene como argumentos al menos una variable polimórfica.
7. Un atributo declarado con visibilidad protegida en una clase A es accesible desde clases definidas en el mismo espacio de nombres donde se definió A.
8. Una de las características básicas de un lenguaje orientado a objetos es que todos los objetos de la misma clase pueden recibir los mismos mensajes.
9. Una operación de clase sólo puede ser invocada mediante objetos constantes.
10. En C++, si no se define ningún constructor, el compilador proporciona por defecto uno sin argumentos.
11. Dada una clase genérica, no se pueden derivar de ella clases genéricas.
12. Una clase interfaz no puede tener instancias.
13. Una clase abstracta siempre tiene como clase base una clase interfaz.
14. No se puede definir un bloque *catch* sin su correspondiente bloque *try*.
15. Tras la ejecución de un bloque *catch*, termina la ejecución del programa.
16. Una variable polimórfica puede hacer referencia a diferentes tipos de objetos en diferentes instantes de tiempo.
17. El *downcasting* estático siempre es seguro.
18. El principio de sustitución implica una coerción entre tipos de una misma jerarquía de clases.
19. La sobrecarga basada en ámbito permite definir el mismo método en dos clases diferentes.
20. Una operación de clase no puede tener enlace dinámico.

FALSO
VERDADERO
FALSO
VERDADERO
FALSO

VERDADERO
FALSO
VERDADERO
FALSO

VERDADERO
FALSO
VERDADERO
FALSO
VERDADERO
FALSO

VERDADERO
FALSO
VERDADERO
VERDADERO
VERDADERO

1. La herencia protegida permite a los métodos de la clase derivada acceder a las propiedades privadas de la clase base.
2. Una clase abstracta se caracteriza por no tener definido ningún constructor.
3. La siguiente clase en C++: `class S {public: virtual ~S()=0;};` define una interfaz.
4. Los métodos virtuales son métodos abstractos.
5. Los métodos abstractos siempre tienen enlace dinámico.
6. Los constructores siempre tienen enlace dinámico.
7. En C++, un atributo de clase debe declararse dentro de la clase con el modificador `static`.
8. Un método sobrecargado es aquel que recibe como argumento al menos una variable polimórfica.
9. Un método tiene polimorfismo puro cuando devuelve una variable polimórfica.
10. Un atributo declarado con visibilidad protegida en una clase A es accesible desde clases definidas en el mismo espacio de nombres donde se definió A.
11. Dada la siguiente definición de clase en C++:

```
class TClase {
  public:
    TClase(int dim);
  private:    int var1;
};
```

    La instrucción `TClase c1;` no da error de compilación e invoca al constructor por defecto.
12. Una de las características básicas de un lenguaje orientado a objetos es que todos los objetos de la misma clase pueden recibir los mismos mensajes.
13. Hablamos de encapsulación cuando agrupamos datos junto con las operaciones que pueden realizarse sobre esos datos.
14. Una operación de clase sólo puede ser invocada mediante objetos constantes.
15. En C++ los constructores se pueden declarar como métodos virtuales.
16. En la misma clase, podemos definir constructores con distinta visibilidad.
17. Si no se captura una excepción lanzada por un método, el programa no advierte que ha ocurrido algún error y continua su ejecución normalmente.
18. En C++, es obligatorio especificar qué excepciones lanza una función mediante una cláusula `throw` tras la declaración de la función.
19. Dada una clase genérica, se pueden derivar de ella clases no genéricas.
20. Una clase interfaz no debe tener atributos de instancia. Una clase abstracta sí puede tenerlos.

1 F	6F	11F	16V
2F	7V	12V	17F
3 V	8F	13V	18F
4F	9F	14F	19V
5V	10F	15F	20V

1. Un mensaje tiene cero o más argumentos. Por el contrario, una llamada a procedimiento/función tiene uno o más argumentos.
2. La interpretación de un mismo mensaje puede variar en función del receptor del mismo y/o del tipo de información adicional que lo acompaña.
3. En una agregación, la existencia de un objeto *parte* depende de la existencia del objeto *todo* que lo contiene.
4. Una forma de mejorar el diseño de un sistema es reducir su acoplamiento y aumentar su cohesión.
5. Los inicializadores en C++ tienen el formato *nombreAtributo(valor)* y se colocan entre la lista de argumentos y el cuerpo de cualquier método, permitiendo asignar valores a los atributos de instancia de la clase en la que se define dicho método.
6. Un atributo estático ocupa una zona de memoria que es compartida por todos los objetos de la clase en la que se define, aunque no por los objetos de cualquier clase derivada de ella.
7. Un atributo de clase debe tener visibilidad pública para poder ser accedido por los objetos de la clase.
8. La herencia múltiple se produce cuando de una misma clase base heredan varias clases derivadas.
9. Un atributo protegido en la clase base es también protegido en cualquier clase que derive de dicha clase base, independientemente del tipo de herencia utilizado.
10. Cuando se crea un objeto de una clase D que deriva de una clase B, el orden de ejecución de los constructores es siempre *B() D()*.
11. Una clase abstracta es una clase que no permite definir instancias de ella.
12. Un interfaz es una clase abstracta con al menos un método de instancia abstracto.
13. Un método de clase (estático) no puede tener enlace dinámico.
14. Un método virtual en C++ siempre tiene enlace estático.
15. El principio de sustitución implica una coerción entre tipos de una misma jerarquía de clases.
16. La llamada a un método sobrescrito se resuelve en tiempo de compilación.
17. El *downcasting* estático siempre es seguro.
18. Los métodos definidos en una clase genérica son a su vez genéricos.
19. No se puede definir un bloque *catch* sin su correspondiente bloque *try*.
20. La instrucción *throw* (en C++) sólo permite lanzar objetos de clase *exception* o de clases derivadas de ella.

1F	6F	11V	16F
2V	7F	12F	17F
3F	8F	13V	18V
4V	9F	14F	19V
5F	10V	15V	20F

1. **En el cuerpo de una operación de clase no se puede acceder a ningún atributo/operación de instancia de los objetos pasados como parámetro**

2. **En una composición un objeto componente puede formar parte de más de un compuesto.**

3. **La interpretación de un mismo mensaje puede variar en función del receptor del mismo y/o de la información adicional que le acompaña.**

4. **Los destructores en C++ pueden aceptar cualquier número de parámetros.**

5. **Los TAD´s representan una perspectiva orientada a datos, mientras que los objetos reflejan una perspectiva orientada a servicios.**

6. **Un atributo estático ocupa una zona de memoria que es compartida por todos los objetos de la clase en la que se define, aunque no por los objetos de cualquier clase derivada de ella.**

7. **Un atributo privado en la clase base no es directamente accesible en la clase derivada, independientemente del tipo de herencia utilizado.**

8. **Un constructor acepta cualquier tipo de modificador (static, const, public, etc)**

9. **Una clase es una especificación abstracta de una estructura de datos y de las operaciones que se pueden realizar con ella.**

10. **Una operación constante sólo puede ser invocada por un objeto constante.**

1F
2F
3V
4F
5V
6F
7V
8F
9F
10F

1. **El puntero this es un puntero constante al objeto que recibe el mensaje**

2. **Implementar la forma canónica ortodoxa de una clase es una condición necesaria (aunque no suficiente) para controlar que el valor de un atributo de clase que cuenta el número de instancias de dicha clase esté siempre en un estado consistente.**

3. **La existencia de una relación todo-parte entre dos clases implica necesariamente que el objeto ´todo´ maneja la creación/destrucción de objetos de tipo ´parte´**

4. **La forma canónica de la clase está formada por el constructor, el constructor de copia, el destructor y el operador de asignación**

5. **Si la clase no proporciona un constructor sin parámetros, el compilador en C++ genera uno de oficio**

6. **Tanto composición como herencia son mecanismos de reutilización del software**

7. **Un atributo de clase debe declararse dentro de la clase con el modificador const**

8. **Un atributo de clase público puede ser accedido desde fuera de la clase a través de un objeto de la clase, un puntero o referencia al mismo o mediante el nombre de la clase seguido del operador de ámbito**

9. **Una clase derivada puede añadir nuevos métodos/atributos propios de la clase derivada, pero no modificar los métodos heredados de la clase base**

10. **Una interfaz es la definición de un protocolo para cierto comportamiento, sin especificar la implementación de dicho comportamiento**

1V
2F
3F
4V
5F
6V
7F
8V
9F
10V

1. **C++ sólo permite heredar cuando la clase hija es un subtipo de la clase padre (herencia como implementación de la generalización)**

2. **El constructor de copia permite argumentos tanto por referencia como por valor.**

3. **El estado de un objeto es el conjunto de valores de atributos y métodos que han sido invocados sobre él.**

4. **En las jerarquías de herencia en C++, si la clase base define un operador de asignación y la clase derivada no lo redefine, al invocar a dicho operador con objetos de la clase derivada se invocará al código de la clase base.**

5. **La herencia es más flexible en cuanto a posibles cambios en la naturaleza de los objetos que la composición**

6. **La herencia privada en C++ es un tipo de herencia insegura porque no preserva el principio de encapsulación.**

7. **La relación de herencia es una relación de clases no persistente.**

8. **Los inicializadores en C++ tienen el formato nombre_atributo(valor) y se colocan entre la lista de argumentos y el cuerpo de cualquier método. Estos inicializadores permiten asignar valores a los atributos de la clase en la que se define dicho método.**

9. **Un objeto se caracteriza por poseer un estado, un comportamiento y una identidad.**

10. **Una clase abstracta siempre tiene que tener alguna clase que derive de ella.**

1F
2F
3F
4F
5F
6F
7F
8F
9V
10F

1. Tanto la herencia protegida como la privada permiten a una clase derivada acceder a las propiedades privadas de la clase base.
2. Una clase abstracta se caracteriza por no tener atributos.
3. La siguiente clase: `class S {public: virtual ~S()=0; virtual void f()=0;};` constituye una interfaz en C++.
4. Desde un método de una clase derivada nunca puede invocarse un método implementado con idéntica signatura de una de sus clases base.
5. Los métodos con enlace dinámico son abstractos.
6. Los constructores de las clases abstractas siempre son métodos abstractos.
7. Un atributo de clase debe tener visibilidad pública para poder ser accedido por los objetos de la clase.
8. Un metodo sobrecargado es aquel que tiene más de una implementación, diferenciando cada una por el ámbito en el que se declara, o por el número, orden y tipo de argumentos que admite.
9. Un método abstracto es un método con polimorfismo puro.
10. Todo espacio de nombres define su propio ámbito, distinto de cualquier otro ámbito.
11. En la sobrecarga de operadores binarios para objetos de una determinada clase, si se sobrecarga como función miembro, el operando de la izquierda siempre es un objeto de la clase.
12. La genericidad se considera una característica opcional de los lenguajes orientados a objetos
13. Hablamos de encapsulación cuando diferenciamos entre interfaz e implementación.
14. Una operación de clase no es una función miembro de la clase.
15. En C++, si no se define ningún constructor, el compilador proporciona por defecto uno sin argumentos.
16. Los constructores siempre deben tener visibilidad pública.
17. En C++, si no se captura una excepción lanzada por un método, se produce un error de compilación.
18. En C++, la cláusula `throw()` tras la declaración de una función indica que ésta no lanza ninguna excepción.
19. Dada una clase genérica, no puede ser utilizada como clase base en herencia múltiple.
20. De una clase interfaz no se pueden crear instancias. De una clase abstracta sí.

FALSO	FALSO	VERDADERO	VERDADERO	VERDADERO	VERDADERO
FALSO	FALSO	FALSO	VERDADERO	FALSO	FALSO
VERDADERO	FALSO	VERDADERO	VERDADERO	FALSO	FALSO
	FALSO		FALSO		

1. Cuando usamos la *varianza* estamos haciendo un uso inseguro de la herencia de implementación.

2. En la sobrecarga de operadores como función miembro, el operando de la izquierda puede ser un objeto de la clase o cualquier otro objeto, mientras que en las funciones amigas siempre es un objeto de la clase.

3. Cuando creamos un objeto en C++ mediante una variable automática el constructor se autoinvoca. También se autoinvoca el destructor del mismo al salir del ámbito de la función donde se creó.

4. Si para una clase genérica llamada Pila<T> declaramos las siguientes funciones:

```
virtual void apilar(T* pt);
virtual void apilar(T t);
```

   Es un caso de sobrescritura ya que el tipo devuelto es el mismo en ambos métodos.

5. De una clase abstracta se pueden crear referencias a objetos de la clase.

6. En un atributo de clase se reserva espacio en memoria para una copia de él por cada objeto de su clase creado.

7. Declarar un dato miembro de una clase como private indica que sólo se puede acceder a ese atributo desde las funciones miembro de la clase.

8. Una clase abstracta siempre tiene que tener alguna clase que derive de ella.

9. El polimorfismo debido a la sobrecarga de funciones siempre se da en relaciones de herencia.

10. A los atributos de instancia si son constantes se les asigna su valor inicial fuera de la clase.

11. Toda sentencia que aparece después de un punto del programa en el que ocurre una excepción, en ningún caso se ejecuta.

12. Si utilizamos los mecanismos de manejo de excepciones disminuye la eficiencia del programa incluso si no se llega a lanzar nunca una excepción.

13. Cuando se captura una excepción y ésta pertenece a una jerarquía de clases, el primer bloque catch debe comenzar con la clase del nivel más alto de la jerarquía.

14. Hablamos de shadowing cuando el método a invocar se decide en tiempo de compilación.

15. A diferencia de otros lenguajes de programación en C++ la sobreescritura en relaciones de herencia se debe indicar de forma explícita en la clase padre .

VERDADERO FALSO FALSO
FALSO FALSO VERDADERO
VERDADERO FALSO FALSO
FALSO FALSO VERDADERO
VERDADERO FALSO VERDADERO

1. La encapsulación es un mecanismo que permite separar de forma estricta interfaz e implementación.
2. La interpretación de un mismo mensaje puede variar en función del receptor del mismo y/o del tipo de información adicional que lo acompaña.
3. Un atributo de clase público puede ser accedido desde fuera de la clase a través de un objeto de la clase, un puntero o referencia al mismo o mediante el nombre de la clase seguido del operador de ámbito.
4. Es posible definir un constructor de copia invocando en su cuerpo al operador de asignación.
5. El recolector de basura es un mecanismo de liberación de recursos presente en todos los lenguajes OO.
6. Para que se pueda realizar una herencia múltiple en C++, es necesario que no coincida ninguno de los nombres de atributo entre las clases base involucradas.
7. Una clase derivada puede añadir nuevos métodos/atributos propios de la clase derivada, pero no modificar los métodos heredados de la clase base.
8. En las jerarquías de herencia en C++, si la clase base define un operador de asignación y la clase derivada no lo redefine, al invocar a dicho operador con objetos de la clase derivada se invocará al método de la clase base.
9. Dada una clase abstracta siempre debe existir alguna clase que derive de ella.
10. La signatura de tipo de un método incluye el tipo devuelto por el método.
11. El principio de sustitución implica una coerción entre tipos de una misma jerarquía de clases.
12. En C++, un destructor no puede ser virtual.
13. El puntero *this* no es una variable polimórfica porque es constante y no se puede cambiar su valor.
14. No se puede derivar una clase no genérica de una genérica.
15. Las instrucciones para el manejo de excepciones nos permiten mezclar el código que describe el funcionamiento normal de un programa con el código encargado del tratamiento de errores.

VERDADERO FALSO FALSO FALSO
VERDADERO FALSO VERDADERO FALSO
VERDADERO FALSO FALSO VERDADERO
VERDADERO FALSO