1. ¿Qué muestra por pantalla el siguiente programa?

```
class Foo{
public:
   Foo(int a):a{a}{}
   virtual int tris(){return a;}
protected:
   int a;
};

class Fii:public Foo{
public:
   Fii(int a):Foo{a}{};
   int tris(){return 2*a;}
};

int main(){
   Fii f{2};
   std::cout << f.tris();
   return 0;
}
```

2
4
Nada, porque presenta errores de compilación
0

Comentado [1]: La clase Foo tiene un método virtual tris que devuelve un tipo entero.

La clase Fii hereda publicamente de Foo y redefine el método tris heredado de su base.

En la función principal instanciamos un objeto Fii con variable miembro a inicializada con el valor 2.

Mostramos por pantalla retorno de tris teniendo en cuenta que la definición de la función en la clase derivada oculta la definición previa en la clase base.

Por lo que mostramos 4

2. ¿Qué muestra por pantalla el siguiente programa?

```
class Foo{
public:
   Foo(int a):a{a}{}
   virtual int tris(){return a;}
protected:
   int a;
};

class Fii:public Foo{
public:
   Fii(int a):Foo{a}{};
   int tris(){return 2*a;}
};

int main(){
   Foo f = Fii{2};
   std::cout << f.tris();
   return 0;
}
```

2
4
Nada, porque presenta errores de compilación
0

Comentado [2]: La clase Foo tiene un método virtual tris que devuelve un tipo entero.

La clase Fii hereda publicamente de Foo y redefine el método tris heredado de su base.

En la función principal instanciamos un objeto Foo con variable miembro a inicializada con el valor 2.

Mostramos por pantalla retorno de tris.
En este caso el objeto es tipo clase base por lo que usamos return a ymostramos 2

3. ¿Qué muestra por pantalla el siguiente programa?

```
class Foo{
public:
   Foo(int a):a{a}{}
   virtual int tris(){return a;}
protected:
   int a;
};

class Fii:public Foo{
public:
   Fii(int a):Foo{a}{};
   int tris(){return 2*a;}
};

int main(){
   std::unique_ptr<Foo> f = std::make_unique<Fii>(2);
   std::cout << f->tris();
   return 0;
}
```

2
4
Nada, porque presenta errores de compilación
0

Comentado [3]: En esta ocasión inicializamos un puntero único de tipo clase base con un puntero a su clase derivada

Y llamamos con el operador -> (por ser un puntero al método tris)

Estaríamos en el caso del ejemplo 1 y mostraríamos 4

4. De los siguientes programas, indica cuáles presentan algún error de compilación.

Los errores no se deben a faltas de puntos y comas, llaves, etc. sino a errores de concepto de programación.

```cpp
// PROGRAMA 1
#include <iostream>
using namespace std;
class Fii{
public:
   Fii(){}
   virtual void foo() = 0;
};

class Fee: public Fii{
public:
   Fee(){}
   void print() const {cout <<
"hola\n";}
};

int main(){
   Fee fu;
   fu.print();
}
```

```cpp
// PROGRAMA 2
#include <iostream>
using namespace std;
class Fii{
public:
   Fii(){}
   virtual void foo() = 0;
};

class Fee: public Fii{
public:
   Fee(){}
   void foo(){  }
   void print() const {cout <<
"hola\n";}
};
int main(){
   Fee fu;
   fu.print();
}
```

```cpp
// PROGRAMA 3
#include <iostream>
using namespace std;
class Fii{
public:
   Fii(){}
   virtual void foo(){ };
};

class Fee: public Fii{
public:
   Fee(){ }
   void print() const {cout <<
"hola\n";}
};
int main(){
   Fee fu;
   fu.print();
}
```

```cpp
// PROGRAMA 4
#include <iostream>
using namespace std;
class Fii{
public:
   Fii(){}
   virtual void foo() = 0 {}
};

class Fee: public Fii{
public:
   Fee(){}
   void foo() {}
   void print() const {cout <<
"hola\n";}
};

int main(){
   Fee fu;
   fu.print();}
```

Comentado [4]: Programa 1: en la clase base tenemos un método virtual puro que no es redefinido en la clase derivada => ERROR

Programa 2: aquí si redefinimos el método virtual puro => OK

Programa 3: en este caso la clase base tiene un método virtual (pero no es puro), por lo que no hay obligación de redefinirlo => OK

Programa 4: en la clase base declaramos un método virtual puro con bloque de código => ERROR
un método virtual puro no tiene bloque, su bloque se define obligatoriamente en las derivadas

1, 4
2, 3
1, 2
2, 4

5. ¿Qué muestra por pantalla el siguiente programa?

```cpp
#include <iostream>
#include <memory>

class Base{
public:
 virtual ~Base(){}
 virtual void Mostrar(){std::cout<< "Soy Base ";}
};

class Der1: public Base{
public:
 void Mostrar(){std::cout<<"Soy Der1 ";}
};

class Der2: public Base{
public:
 void Mostrar(){std::cout<<"Soy Der2 ";}
};

int main(){
 std::unique_ptr<Base> p1 = std::make_unique<Der1>();
 auto p2= std::make_unique<Der2>();

 p1->Mostrar();
 p2->Mostrar();

 return 0;
}
```

Soy Der1 Soy Der2
Soy Base Soy Der1 Soy Der2
Soy Der1 Soy Der2 Soy Base
No sé quien soy ya

Comentado [5]: Tenemos una clase base y 2 clases derivadas que heredan de ella

La base tiene un método virtual Mostrar con el que dice quien es y las derivadas lo redefinen con su nombre

En la función principal en un puntero base inicializamos con un puntero a derivada 1

Y en otro puntero a base inicializamos con un puntero a derivada 2

Al llamar a Mostrar() de cada uno de los punteros estaremos usando el mostrar de las derivadas, ya que el método virtual esta redefinido (si no lo estuviera usaría el método virtual de la base)

6. Para la siguiente clase base indica qué clases derivadas presentan algún error de compilación.

```
class Padre{
public:
   Padre(int pub,int pri, int pro){Pub=pub;Pri=pri;Pro=pro;}
   int Pub;

private:
   int Pri;

protected:
   int Pro;
};
```

Los errores no se deben a faltas de puntos y comas, llaves, etc. sino a errores de concepto de programación.

```
// PROGRAMA 1
#include <iostream>
using namespace std;


class hPub:public Padre{
public:
   hPub():Padre(5,55,555){}
   void foo(){cout<<Pri<<:endl;}
};
```

```
// PROGRAMA 2
#include <iostream>
using namespace std;


class hPri:private Padre{
public:
   hPri():Padre(6,66,666){}
   void foo(){std::cout<<Pub<<endl;}
};
```

```
// PROGRAMA 3
#include <iostream>
using namespace std;

class hPro :protected Padre{
public:
   hPro():Padre(7,77,777){}
   void foo(){cout<<Pro<<endl;}
   //void
};
```

```
// PROGRAMA 4
#include <iostream>
using namespace std;

class hPri2:private Padre{
public:
   hPri2():Padre(6,66,666){}
   void foo(){std::cout<<Pro<<endl;}
};
```

Programa 4
Programa 3
Programa 2

Comentado [6]: Tenemos una clase Padre con un miembro publico, otro privado y otro protegido.

Programa 1: heredamos públicamente e intentamos acceder a la parte privada => ERROR

Programa 2: heredamos privadamente y accedemos a la parte pública => OK

Programa 3: heredamos protegidamente y accedemos a la parte protegida => OK

Programa4: heredamos privadamente y accedemos a la parte protegida => OK

Programa 1

7. ¿Qué muestra por consola el siguiente programa?

```
#include <iostream>

int main() {
   int *a = new int;
   int *b = a;
   int *c = new int;
   *a = 5;
   *b = 6;
   *c = 7;
   *b = *a;
   std::cout << *b << "\n";

   return 0;
}
```

Comentado [7]: Tenemos un puntero a int inicializado con reserva dinámica de memoria

Tenemos un puntero b que apunta donde a

Y un puntero c con reserva dinámica de memoria

Asignamos al contenido de a 5 (*a y *b tendran 5)
Al contenido de b 6 (*a y *b tendran 6)
Al contenido de c 7

Al contenido de b asignamos el contenido de a (continúan siendo 6)

Mostramos el contenido de b => 6

5
6
7
Una dirección de memoria

8.- ¿Qué muestra por consola el siguiente programa?

Comentado [8]: Tenemos una base y una derivada que saludan con su nombre

Creamos 2 punteros uno a base y otro a derivada.

Los inicializamos con reserva dinámica de memoria

Y llamamos a los saludos. Como Saluda() en la base no es virtual => Mostramos las 2 veces Soy Base

Si Saluda fuera virtual => Soy Base Soy Derivada

```cpp
#include <iostream>

class Base
{
public:
    void Saluda(){std::cout<<"Soy Base";}
};

class Der1:public Base
{
public:
    void Saluda(){std::cout<<"Soy Derivada 1 ";}
};

int main()
{
    Base *miBase,*miDer1;

    miBase=new Base;
    miDer1=new Der1;

    miBase->Saluda();
    miDer1->Saluda();

    return 0;
}
```

Soy Derivada 1 Soy Derivada 1
Soy Derivada 1 Soy Base
Soy Base Soy Derivada 1
Soy Base Soy Base

9.- ¿Qué muestra por consola el siguiente programa?

Comentado [9]: En este caso Saluda es virtual por lo que mostramos Soy Base Soy Derivada

```cpp
#include <iostream>
```

```
class Base
{

public:
    virtual void Saluda(){std::cout<<"Soy Base ";}
};

class Der1:public Base
{
public:
    void Saluda(){std::cout<<"Soy Derivada 1 ";}
};

int main()
{
    Base *miBase,*miDer1;

    miBase=new Base;
    miDer1=new Der1;

    miBase->Saluda();
    miDer1->Saluda();

    return 0;
}
```

Soy Derivada 1 Soy Derivada 1
Soy Derivada 1 Soy Base
Soy Base Soy Derivada 1
Soy Base Soy Base

10.- ¿Qué muestra por consola el siguiente programa?

Comentado [10]: Este caso daría error de compilación ya que Base es una clase abstracta por tener un método virtual puro y por tanto no podríamos instanciar objetos de su tipo

```
#include <iostream>
```

```
class Base
{
public:
    virtual void Saluda()=0;
};

class Der1:public Base
{
public:
    void Saluda(){std::cout<<"Soy Derivada 1"<<std::endl;}
};

int main()
{
    Base *miBase,*miDer1;

    miBase=new Base;
    miDer1=new Der1;

    miBase->Saluda();
    miDer1->Saluda();

    return 0;
}
```

0
Soy Derivada 1
0 Soy Derivada 1
Nada, porque presenta error de compilación