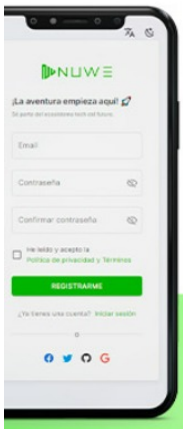
TareasTema4.pdf

Anónimo

Programación Orientada a Objetos

2º Grado en Ingeniería Informática

Escuela Superior de Ingeniería
Universidad de Cádiz



TAREAS 4.1

Exercise 1
Is the following program correct (i.e., will this program compile)?

```
1 #include <iostream>
3 using namespace std;
5 void print(int i) { cout << i << " [integer]" << endl; }
6 void print(float f) { cout << f << " [float]" << endl; }
8 int main()
9   {
10   print(2); //imprime 2 (llamada a print(int i))
11   print(2.0); //error, llamada ambigua a print(double)
12   print('a'); //imprime 97, conversión de char a int (llamada a print(int i)
13 }
```

Itś not correct because in line 11, the compiler doesnt know whos function to call with a double number. (Ambas son válidas, de double a float y a int)

Exercise 2
Given the following overloaded functions, which of the calls to these functions (lines 7 to 10) are allowed?

```
1 void f(char);
2 void f(double);
3 void f(void *);

5 int main()
6 {
7     f(0.0); //llamada a f(double)
8     f(0); //error, llamada a f(int) es ambigua
9     f(0.0f); ////llamada a f(double)
10   f("Hello"); //error, no hay función para llamar f(const char [6])
11 }            //conversión inválida de const char [6] a char
                 //conversión inválida de const void* a void*
```

Exercise 3
Consider a base class B and a derived class D . Both classes define a member function f() . Justify whether the following code snippet is correct and, if so, indicate which member function f() would be called, considering two independent cases:

1 when B::f() is virtual and
2 when B::f() is not virtual.

```
1 B b, *bp;
2 D d, *dp;

4 bp = &d;
5 bp→f(); // 1. llama a D::f() y 2. llama a B::f()

6 dp = &b; //error, conversión hacia abajo
7 dp→f(); // no válida
```

```
8 dp = &d;
9 dp→f(); // 1. llama a D::f() y 2. llama a D::f()
```

Exercise 4
Indicate what the following program will exactly send to the standard output when executed:

```
1 #include <iostream>

3 struct B {
4     B() { std::cout << "Constructor (class B)\n"; }
5     virtual ~B() { std::cout << "Destructor (class B)\n"; }
6 };

8 struct D: B {
9     D() { std::cout << "Constructor (class D)\n"; }
10   ~D() { std::cout << "Destructor (class D)\n"; }
11 };

13 int main() {
14   B *pb = new D;
15   delete pb;
16 }
```

```
Constructor (class B)
Constructor (class D)

Destructor (class D)
Destructor (class B)
```

Para eliminar un objeto derivado D, debemos llamar a los dos destructores ~D y ~B.
Si creo un objeto de tipo D, llama al contructor de B y de D.

Siendo ~B virtual, delete pb llama al destructor de D y luego de B, si no, solo llama al destructor de B.

What would it happen if we removed the keyword virtual?
Si ~B no es virtual el programa LLAMARA AL DESTRUCTOR DE B pero la parte correspondiente a D se queda sin destruir.

Exercise 5
Which of the destructors in these classes are virtual?

```
class A { public: ~A(); };
class B : public A { public: virtual ~B(); };
class C : public B { public: virtual ~C(); };
class D : public C { public: ~D(); };
```

~B, ~C y ~D. Ya que ~B es virtual, toda su descendencia tendrá destructor virtual.
Los constructores no pueden ser virtuales.

NUWE

Encuentra el trabajo de tus sueños

Participa en retos y competiciones de programación

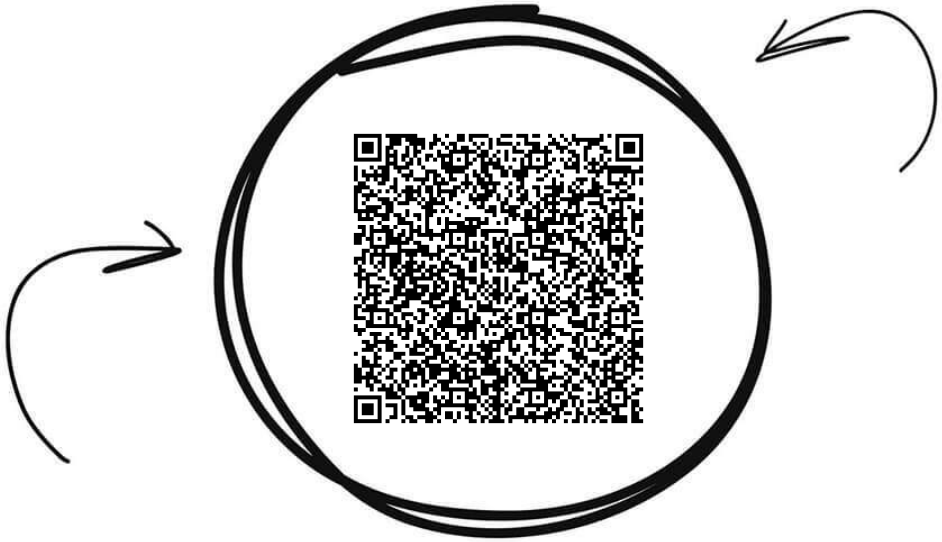
Ten contacto de calidad con empresas líderes en el sector tecnológico mientras vives una experiencia divertida y enriquecedora durante el proceso.

Únete ahora

Escanéame y obtén más info!!

hello@nuwe.io

Programación Orientada a Obj...



Banco de apuntes de la

WUOLAH

Comparte estos flyers en tu clase y consigue más dinero y recompensas

1. Imprime esta hoja
2. Recorta por la mitad
3. Coloca en un lugar visible para que tus compis puedan escanar y acceder a apuntes
4. Llévate dinero por cada descarga de los documentos descargados a través de tu QR

Exercise 6
If we removed the member function area() from the class square , would the behaviour of this class change?

```
class rectangle {
public:
    rectangle(double w, double l): width(w), length(l) {}
    virtual double area() { return width * length; }
private:
    double width, length;
};
class square: public rectangle {
public:
    square (double l): rectangle(l, l) {}
    double area() { return rectangle::area(); }
};
```

No cambiaría nada, ya que rectangle::area() es heredado por rectangle.

TAREAS 4.2

Exercise 1
Consider a base class B and a derived class D . Both classes have a member function f() , which is defined as pure virtual in class B , so D::f() overrides B::f().

1. **Write the definition of B::f() (it takes no parameters and it returns no value).**
   ```
   Class B {
       public:
           virtual void f() = 0;
    };
   ```

2. **What is the kind of class B denominated?**
   Base class and abstract class due to having a pure virtual method.

3. **What happens when the below code is compiled and executed? Is there a static (at compile-time) or dynamic (at runtime) binding in line 4?**

   ```
   1 B b, *bp;
   2 D d;
   3 bp = &d;
   4 bp→f();
   ```

   It calls to D::f() and it's a dynamic binding.
   En una clase abstracta no se pueden definir objetos, pero sí punteros

Exercise 2
Consider the following class hierarchy, in which X , Y and Z inherit publicly from V :

```
struct V {
    virtual void fv() = 0;
    virtual ~V() {}
```

```
};

struct X : V {
    void fv() {}
};

struct Y : V {
    void fv() {}
};

struct Z : V {
    void fv() {}
};
```

Think about a function f() that processes objects of class V :

```
void f(V& v)
{
    if (typeid(v) == typeid(X)) {
        std::cout << "Processing object X...\n";
        // specific code for X
    }

    if (typeid(v) == typeid(Y)) {
        std::cout << "Processing object Y...\n";
        // specific code for Y
    }

    if (typeid(v) == typeid(Z)) {
        std::cout << "Processing object Z...\n";
        // specific code for Z
    }
}
```

1. **What is the output of the following instructions?**

```
X x;
V* pv = new Y;
f(x);
f(*pv);
```

Processing object X…
Processing object Y…
(Comprobado en CodeBlocks)

2. **Is this the best way to implement the polymorphic behavior of f() ? Justify your answer. Depending on your answer, describe how this implementation can be improved and, if necessary, modify the code to get the same output.**

No, debido a que necesita que el método se declare para cada clase de forma que no haya ambigüedades. Para mejorarlo podríamos hacer:

```
struct V {
    virtual void fv() = 0;
    virtual ~V() {}
};

struct X : V {
    void fv() {std::cout << "Processing object X...\n";}
};

struct Y : V {
    void fv() {std::cout << "Processing object Y...\n";}
};

struct Z : V {
    void fv() {std::cout << "Processing object Z...\n";}
};
```

Y para obtener el mismo resultado modificaríamos el código aśi:

```
X x;
V* pv = new Y;
x.fv();
pv→fv();
```

Exercise 3

1. **Define a class template Buffer to represent a storage area for items of the same type. Its template parameters are:**

- the type of the stored items (by default, the type of size 1 byte);
- the storage capacity (by default, 256 items).

Define the Buffer ’s main data member (based on the STL container vector) and the default constructor.

```
template <typename T = byte, size_t n = 250>
class Buffer {
    public:
        Buffer() : vec(n);
    private:
        std::vector<T> vec;
};
```

2. **Then, define an object of type Buffer composed of 128 items of type int , and another one composed of 256 items of the default type.**

```
Buffer<int, 128> a;
```

Buffer<> b;

Exercise 4

1. **Write the output of the following program.**

1BILi0EE::1BILi0EE() //creación objeto b de tipo B<0>
1BILi1EE::1BILi1EE() //creación objeto b1 de tipo B<1>
1BILi2EE::1BILi2EE() //creación objeto b1 de tipo B<2>
D::D() //creación objeto de tipo D derivado de b

1BILi0EE::1BILi0EE(const 1BILi0EE&) //creación objeto b de tipo B<0> a partir de otro
1BILi1EE::1BILi1EE(const 1BILi1EE&)
1BILi2EE::1BILi2EE(const 1BILi2EE&)
D::D(const D&)

D::~D() //destrucción objeto D::d
1BILi2EE::~1BILi2EE() //destrucción objeto D::b2 de tipo B<2>
1BILi1EE::~1BILi1EE() //destrucción objeto D::b1 de tipo B<1>
1BILi0EE::~1BILi0EE() //destrucción objeto B::b de tipo B<0>

D::~D()
1BILi2EE::~1BILi2EE()
1BILi1EE::~1BILi1EE()
1BILi0EE::~1BILi0EE()

// ¿Por qué repite otra vez el mismo proceso?
Este último bloque se produce al terminar el programa, el compilador destruye automáticamente el objeto d, creado a partir del objeto b.

2. **If classes B<id> were non-polymorphic types, what would be the result?**

Si quitamos el virtual, debemos cambiar también dynamic_cast por static_cast, debido a que este operador sólo se puede usar con tipos polimórficos. Al dejar de ser polimórfica la clase B<0> (cosa que ocurre al retirar la declaración virtual del destructor) se produce un error.

Una vez corregido, el resultado es este:

1BILi0EE::1BILi0EE()
1BILi1EE::1BILi1EE()
1BILi2EE::1BILi2EE()
D::D()

1BILi0EE::1BILi0EE(const 1BILi0EE&)
1BILi1EE::1BILi1EE(const 1BILi1EE&)
1BILi2EE::1BILi2EE(const 1BILi2EE&)
D::D(const D&)

1BILi0EE::~1BILi0EE()

D::~D()
1BILi2EE::~1BILi2EE()

```
1BILi1EE::~1BILi1EE()
1BILi0EE::~1BILi0EE()

int main() {
   B<0>& b{*new D}; //conv hacia arriba
   cout << "-----------------------" << endl;
   D d{static_cast<D&>(b)}; //conversión hacia abajo
   cout << "-----------------------" << endl;
   delete &b; //llama a destructor de D porque &b apunta a D y ~B() es virtual
   cout << "-----------------------" << endl;
}
```