# TareasTema2.pdf

Anónimo

Programación Orientada a Objetos

2º Grado en Ingeniería Informática

Escuela Superior de Ingeniería
Universidad de Cádiz

# SEMINARIO 2 - Tarea 2.1

**Ejercicio 1**
***¿Cómo implementa el lenguaje de programación C++ el principio de encapsulamiento?***
A través de las clases, las cuales agrupan bajo en el mismo nivel de abstracción todos los objetos con iguales características y comportamiento.

**Ejercicio 2**
***¿Cómo implementa el lenguaje de programación C++ el principio de ocultación de información?***
Con la sección pública (interfaz) y la privada (detalles de implementación) de las clases.

**Ejercicio 3**
***¿Hay algún error en el siguiente programa? Si es así, explique por qué y corríjalo.***

```
#include <iostream>
class C {
public:
   C(int i = 0): n(i) {}
   void mostrar() { std::cout << "i = " << n << std::endl; }
private:
   int n;
};

int main() {
   const C c;
   c.mostrar(); //1
}
```

1. No se puede aplicar una función no const a un objeto const, seria void mostrar() const {...}
Si fuera al revés, una función const y un objeto no const, no pasaría nada.

**Ejercicio 4**
***Indique los errores que hay en el siguiente código y su causa.***

```
class C {
public:
   C();
   C(int a, int b, int c, int d);
   int f1(int i) const;
   int f2(int i);
   static void f3() {m = l;} //1.
   static int n;
private:
   mutable int i;
   const int j;
   mutable int k; //se puede modificar aunque la funcion que lo haga sea const
   int l;
   static int m;
};
```

```
int C::f1(int i) const
{
   l = i;  // l no es mutable y la función es const, error al modificar l.
   k = i;
   return 0;
}

C::C() {i = j = k = l = 0;} //2

C::C(int a, int b, int c, int d) : i(a), j(b), k(c), l(d) {}

int C::f2(int i)
{
   k = i;
   l = i;
   return 0;
}
```

1. Los métodos static no existen individualmente para cada objeto de la clase, sólo existe uno para todos. int l existe para cada objeto de la clase, no se puede asignar a un miembro de clase con una función general. Una función static no sabe a qué objeto se refiere ese atributo l.

2. Dos errores, primero no inicializar j y segundo, asignarle su valor a una variable.

Problema futuro: m y n son static y se declara pero no se inicializan, no da problemas de compilación.

**Ejercicio 5**
***Considerando el código de la pregunta anterior, determine si hay errores, en tal caso corríjalos, e indique qué imprime el programa.***

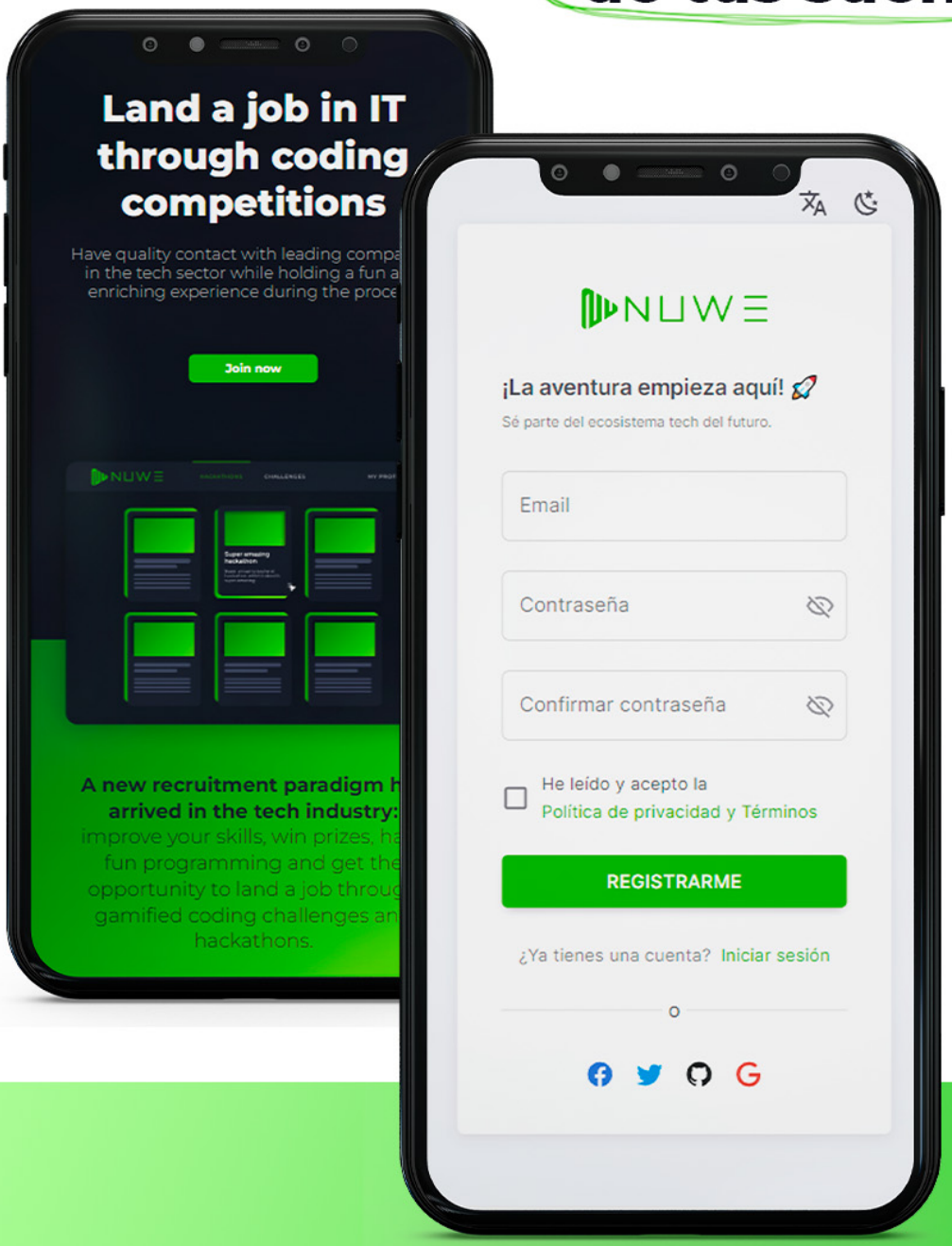
```
#include <iostream>
class C { // clase C anterior
// ...
};
// métodos de la clase C aquí ...

int main()
{
   C c;
   C::n = 3; //Asignación de atributo estatico. Hay que inicializarlo antes (correcto)
   c.n = 4; //n es general para todos los objetos, no único de c (correcto)
   std::cout << C::n << " " << c.n << std::endl;
   return 0;
}
```

Sólo es necesario inicializar los atributos estáticos antes de su uso para que el código sea correcto. Lo mismo es asignar valor a C::n que a c.n, ambas modifican un único atributo de clase.

NUWE

# Encuentra el trabajo de tus sueños



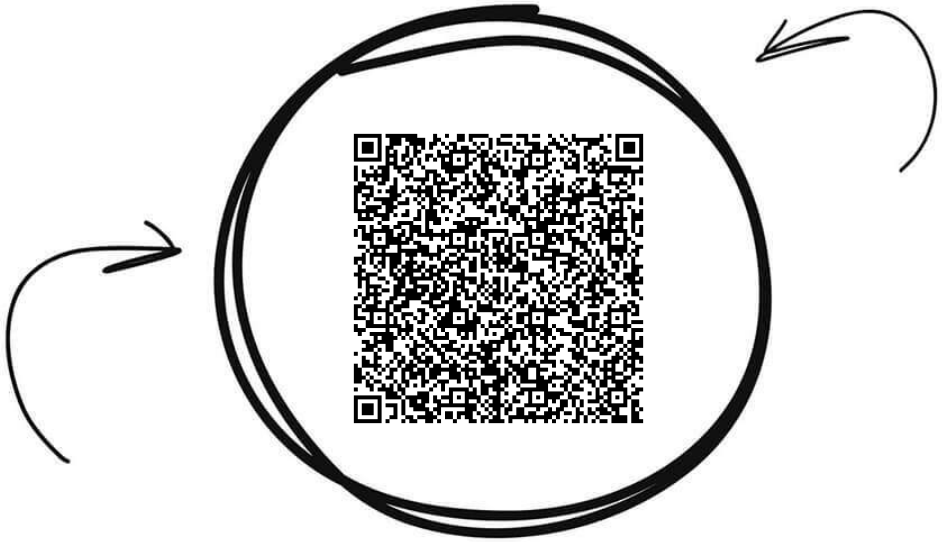

Escanéame y obtén más info!!

**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**

# 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

**Ejercicio 6**
***Considere la siguiente clase:***

```
class C {
public:
   C(int i) : n(i), m(0.0) {} // ctor. de conversión
   // ...
private:
   int n;
   double m;
};
```

**Escriba las declaraciones correspondientes a la sobrecarga del operador + para objetos de tipo C como miembro y como función externa. A continuación escriba a modo de ejemplo un trozo de código que provoque un error de compilación con el operador miembro, pero no con el externo.**

Miembro: C& operator +(const C& c1);
Externo: C operator+ (const C &c1, const C &c2);

```
C c1, c2;
c1 = 5 + c2;
```

Los operadores internos no permiten la conversión implícita a C, mientras que los externos sí.

# SEMINARIO 2 - Tarea 2.2

**Ejercicio 1**
***¿Se puede definir un constructor con un nombre diferente al de la clase?*** **Justifique la respuesta.**

No, porque el compilador va a buscar ese nombre.
Las reglas especifican que los constructores han de tener el mismo nombre de la clase de la cual construye objetos.
Justificación: si no fuera así, y construyera objetos utilizando métodos, el constructor no podría ser sustituido por algún método en muchas ocasiones.

**Ejercicio 2**
***Enumere las diferencias existentes entre inicializar un atributo en la lista de inicialización y asignarle un valor en el cuerpo del constructor. ¿Es posible utilizar dicha lista en otros métodos de una clase?***

Inicializar un atributo en la lista de inicialización significa crear el objeto y asignarle un valor, en cambio, asignar un valor en el cuerpo del constructor requiere que el objeto haya sido inicializado ya con un valor por defecto que se machacará.

Inicializar es, en el mismo acto en que se crea el objeto (se reserva memoria), le das un valor.
Asignar es para un objeto que ya ha sido creado (que tendrá un valor (basura)) le damos un valor nosotros.

Para asignar, primero debes inicializar.

No es posible utilizar dicha lista en otros métodos de una clase, sólo se permite en los constructores, los métodos no inicializan variables.

No confundir con lista inicializadora (std::initializer_list).

**Ejercicio 3**

```
1 class punto {
2    double x, y;
3  public:
4    punto(double a = 0., double b = 0.) : x{a}, y{b} {} //ctor por omisión, 2 parámetros y 1 sólo
     parámetro(conversión)
5    punto(const punto &p) : x{p.x}, y{p.y} {} //ctror de copia
6    punto& operator =(const punto &p) //operador de asignación
7    { x = p.x; y = p.y; return *this; }
8  };
```

**Diga qué función de la clase punto se llama en cada una de las siguientes líneas, o si es incorrecta, suponiendo que se han ejecutado las anteriores, corregidas si es necesario.**

**1) punto p;**
llamada al ctror por defecto
**2) punto q();**
declaración de función q que devuelve objeto de tipo punto
**3) punto r(2.,);**
error de sintaxis, no puedes poner coma y dejar el parámetro vacío, tampoco puedo omitir parámetros a la derecha.
**4) punto s{3.4};**
llamada al constructor de 1 parámetro, omite el segundo.
**5) punto t{};**
llamada al ctror predeterminado
**6) punto u(q);**
suponiendo que q fuera un Punto, llamaría al ctror de copia.
**7) punto v = r;**
inicialización de un objeto v de tipo Punto, llamada al ctror de copia, no se llama al operador de asignación pq es una inicialización.
**9) punto v; v = r;**
primero al constructor predeterminado, luego al operador de asignación.
**8) t = s;**
Asignación, llamada al operador de asignación.

**Ejercicio 4**
***Dadas las clases Libro1 y Libro2***

```
class Libro1 {
   string titulo_; int pags_;
public:
   Libro1(string t="", int p=0); //2 pará,etros omisibles
```

```
   // ...
};

class Libro2 {
   string titulo_; int pags_;
public:
   Libro2(string t, int p=0);
   Libro2(const char* c);
   // ...
};
```

***decida si X se puede sustituir por 1 o 2 en los siguientes items:***

**1. Se puede definir: LibroX lib1;**
Libro1 lib1; //llamada al ctro pred. de Libro1
Libro2 lib2; //llamada al ctro pred. de Libro2 (no, porque no tiene) -> error

**2. Se tiene un constructor de conversión de std::string a LibroX**
std::string a Libro1 //llamada ctror pred. actuando de conversión al tener valor por defecto para p
std::string a Libro2 //llamada al ctror de conversión

**3. Se puede definir: LibroX lib2[5];**
Libro1 lib2[5]; //vector de 5 objetos de tipo Libro1 creados por defcto, para ello necesito crear los objetos individuales, y para ello necesito un ctror predeterminado
Como tiene ctror pred. correcto.
Libro2 lib2[5]; //como no tiene ctror pred. no se puede, incorrecto
El ctror por omisión no se crea automáticamente si el usuario ya ha creado algún ctror para dicha clase.

**4. Se puede definir: std::vector<LibroX> lib3;**
std::vector<Libro1> lib3; //correcto
std::vector<Libro2> lib3; //correcto, con std::vector solo se crea el vector, no se inicializa

**5. Se produce una conversión implícita de const char* a string al ejecutar LibroX* lib4 = new LibroX("El Quijote");**
Libro1* lib4 = new Libro1("El Quijote"); //correcto dado que string tiene ctror de conversion a const char *
Libro2* lib4 = new Libro2("El Quijote"); //no se produce conversión porque ya tiene ctor de conversion desde const char*

**6. Se puede definir: LibroX lib5 = "El Quijote";**
Libro1 lib5 = "El Quijote"; //incorrecto al no tener ctor de conversión
Libro2 lib5 = "El Quijote"; //correcto al tener ctor de conversión

**EJERCICIO 5**

```
#include <iostream>
#include <cstring>
using namespace std;

class Libro {
```

```
    char* titulo_; int paginas_;
  public:
    Libro() : titulo_(new char[1]), paginas_(0) {*titulo_= 0;} //ctror

    Libro(const char* t, int p) : paginas_(p) {
      titulo_ = new char[strlen(t) + 1];
      strcpy(titulo_, t);
    }

    ~Libro() { delete[] titulo_; }
    void paginas(int p) { paginas_ = p; }
    int paginas() const { return paginas_; }
    char* titulo() const { return titulo_; }
};

void mostrar(Libro l) {
  cout << l.titulo() << "␣tiene␣" << l.paginas() << "␣páginas" << endl;
}

int main() {
  Libro l1("Fundamentos␣de␣C++", 474), l2("Por␣Fin:␣C␣ISO", 224), l3;
  l3 = l1; //error, no hemos sobrecargado el operador de asignacion
  mostrar(l1), mostrar(l2), mostrar(l3); //fallo, tenemos que usar punto y coma.
}
```

***Diga si el programa funciona correctamente. En caso afirmativo indique lo que imprime. En caso negativo haga las modificaciones necesarias para que funcione correctamente.***

Operador de asignación para Libro:

```
Libro& operator =(const Libro& l)
{
    if(this != &l) //evitar autoasignación
    {
        paginas_ = l.paginas_;
        delete[] titulo_;
        titulo_ = new char [strlen(l.titulo_)+1];
        strcpy(titulo_, l.titulo);
    }
    return *this;
}
```

Constructor de copia para Libro:

```
Libro (const Libro &l) : paginas_(l.paginas)
{
    titulo_ = new char [strlen(l.titulo_)+1];
    strcpy(titulo_, l.titulo);
}
```

**EJERCICIO 6**
***¿Se consigue llamar a la función f o se producirá un error? En tal caso, corríjalo.***

```
1 struct B;
2 struct A {
3     A(B);
4     // …
5 }
6 struct B {
7     operator A();
8 // …
9 };

11 void f(A&);

13 int main() { B b; f(b); }
```

Existe un error de ambigüedad porque el compilador intenta convertir un objeto de tipo B a uno de tipo A y no sabe con qué función hacerlo de entre dos opciones:

- Constructor de conversión de B a A de la clase A.
- Operador de conversión de B a A de la clase B.

# SEMINARIO 2 - Tarea 2.3

**Ejercicio 1**
***Clasifique las funciones y operadores miembro de la clase matriz en diferentes categorías (constructores, destructores, observadores y modificadores).***

¿Por qué no hay destructor?
Si el usuario no define uno, el compilador crea uno predeterminado, por lo que sí lo hay.

**Constructores**:
explicit matriz(size_t m = 1, size_t n = 1, double y = 0.0)
//Ctor de 3, 2, 1 parámetros, por defecto y conversión (explicit)

matriz(size_t m, size_t n, double f(size_t i, size_t j));
//Ctor de 3 parámetros: m, n y f(función que devuelve un double)

matriz(const initializer_list<valarray<double>>& l); // C++11
//Ctor de conversión desde lista inicializadora de doubles.

matriz(const matriz&) = default; //Ctor de copia por defecto, recibe una ref a un objeto de la misma clase

matriz(matriz&&) = default; //Ctor de movimiento por defecto

**Observadores:**
size_t filas() const; //Obtener filas
size_t columnas() const; //Obtener columnas

double operator ()(size_t i, size_t j) const; //Operador () selección elemento
valarray<double> operator [](size_t i) const; //Operador [] selección de fila
valarray<double> operator ()(size_t j) const; //Operador () selección de columna

(Todo lo que tenga const es observador)

**Modificadores:**
- default: no lo vamos a implementar nosotros y queremos que lo haga el compilador
- todos los operadores de asignación son modificadores

matriz& operator =(const matriz&) = default; //Operador de asignación por copia
matriz& operator =(matriz&&) = default //Operador de asignación por movimiento

double& operator ()(size_t i, size_t j); //Operador () selección elemento
slice_array<double> operator [](size_t i); //Operador [] selección de fila
slice_array<double> operator ()(size_t j); //Operador () selección de columna
matriz& operator =(double y); //Operador de asignación de valor constante

matriz& operator +=(const matriz& a); //Operador +=
matriz& operator -=(const matriz& a); //Operador -=
matriz& operator *=(const matriz& a); //Operador *=
matriz& operator *=(double y); //Operador *= con valor constante

//Son no const para poder asignar, por eso devuelve una referencia

**Otros:**
friend matriz operator -(const matriz& a); //Operador – matriz inversa
//Al ser friend, no es de la clase, no es observador ni modificador

using std::valarray; //Vector de valores numéricos

**Ejercicio 2**
***Si existen los siguientes constructores, escriba una instrucción en cada caso en la que se invoque al mismo y diga si se utiliza en el programa de prueba (en caso afirmativo indique dónde).***

*Constructor predeterminado:* matriz A; → no se utiliza en el main()
*Constructor de copia:* matriz A{B}; //B es una matriz definida anteriormente
matriz Z = matriz(5); → utilizado en main()
*Constructor de movimiento:* matriz A(std::move(B)) → no se utiliza en main()
*Constructor de conversión:* matriz Z = {{1,2}, {5,6}};
matriz n(2);
matriz D = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}; → se utiliza en main()

**Ejercicio 3**

***Describa los errores que hay en el siguiente código:***

1 **matriz A = 3;** //ERROR
2 **matriz B = matriz(5);** //op de asig, matriz de 5x1 con el valor 0
3 **matriz C(3);** //Llamada a ctror de conversión (?)
4 **B = 2;** //B.operator = (2)
5 **A = matriz(5);** //Asignación de matriz a matriz (de movimiento)
6 **B = static_cast<matriz>(4);** //igual que B = matriz(4)

```
matriz A = 3;                 //Op. de asig. desde int -> no definido (error)
matriz B = matriz(5);         //Ctor de conv desde size_t y ctor de copia (op. asig.?)
matriz C(3);                  //Ctor de conv desde size_t
B = 2;                        //Op. de asig. desde int a size_t a matriz
A = matriz(5);                //Ctor de conv desde size_t y op. asig. por copia (asig. por mov.?)
B = static_cast<matriz>(4);   //Ctor de conv desde size_t y op. asig. por copia (asig. por mov.?)
```

***¿Por qué se declara explicit el primer constructor de la clase matriz ? ¿Podría causar algún problema si no se hiciera así? Y en caso afirmativo, ¿se podría evitar ese problema definiendo un operador de conversión de matriz a int? Razone la respuesta.***

El error se debe a que el constructor de conversión es explicit, lo que significa que no se puede llamar de forma implícita. El operador de asignación requiere otra matriz como parámetro, y como 3 no puede ser convertido implícitamente a una matriz, dará un error en tiempo de compilación. Para hacer conversiones de size_t/int a matriz debemos hacerlo de las formas en las líneas 2, 3 y 6. En la línea 4 se llama al operador de asignación a valor constante, así que no es equivalente.

El constructor está explícitamente hecho para evitar conversiones de enteros a matriz. Si no fuera explícito, cualquier entero sería convertido erróneamente a matriz cada vez que una operación lo necesitara.

Matemáticamente no es posible sumar un entero y un array, cualquier forma de conversión implícita causaría errores en el programa y no sería consistente.

No se arregla porque el operador de conversión es diferente del ctror de conversión.

**Ejercicio 4**
***¿Por qué devuelven tipos distintos los operadores de signo + y - ?***

```
const matriz& operator +(const matriz& a);
friend matriz operator -(const matriz& a);
```

El operador + no hace ninguna modificación a la matriz, por eso devuelve una referencia constante al objeto.
Sin embargo, el operador – realiza un cambio de signo de sus elementos y devuelve la matriz por valor, que activa el operador de movimiento. Esto es así porque las variables locales creadas en métodos son eliminadas cuando salen de su ámbito, si fuera devuelta por referencia su comportamiento sería indeterminado por haber sido eliminada.

En el operador + podemos permitirnos devolver la matriz por referencia constante, pues no ha sido creada en la función, sino externa, y no será eliminada al salir de la misma.

En el operador – no podemos devolver por referencia, ya que al salir de la función el objeto sería eliminado. Debemos devolver por valor para activar el operador de movimiento y pasar el valor de una variable interna a la función a una externa.

**Ejercicio 5**
***¿El operador - de cambio de signo es miembro de la clase matriz?***
***¿Se podría definir de la otra forma? En caso afirmativo, escriba la declaración. ¿Qué ventajas e inconvenientes tendría?***

No, es friend, lo que significa que no es miembro de la clase matriz pero sí puede acceder a los atributos privados y protegidos de la misma.
Sí, declarándola miembro de la función y sin parámetros, dentro de la misma utilizaríamos el objeto this.

```
matriz& operator -();

inline matriz& matriz::operator -(){
   x = -x;
   return *this;
}
```

Ventajas:
Inconvenientes:

**Ejercicio 6**
***¿Es correcto definir el operador *= como sigue?***

```
1 inline matriz& matriz::operator *=(const matriz& a)
2 {
3    n = a.columnas();
4    x *= a.x;
5    return *this;
6 }
```

Es correcto ya que no da errores, pero no hace lo que debe, no multiplica dos matrices.
Multiplica uno a uno sus elementos.

Lo correcto sería:

**PRACTICA P1**

```
vector<int>::iterator p = vec.begin(); //Creacion de un iterador
vec.begin(); //acceso al principio del vector
vector<int>::iterator p; //p es un tipo iterator que está definido en la clase vector

Cadena::iterator p = cad.begin();
cout<< *p <<endl;
```

```
for (Cadena::iterator i = a.begin(); i != a.end(); ++i)
      (*i)++;
```

i es un puntero a char
definir un iterador de tipo char*

make fecha.o
make cadena.o
make test-fechacadena-consola.o
g++ ~~c std=c++17 testfechacadena~~consola.cpp
g++ ~~c std=c++17 testfechacadena~~consola.cpp

test-P0-auto: test-caso0-fecha-auto.o test-caso0-cadena-auto.o test-auto.o $(OBJETOS) $(CXX) $(CXXFLAGS) $^ -o $@ test-caso0-fecha-auto.o test-caso0-cadena-auto.o test-auto.o: test-caso0-

fecha-auto.cpp test-caso0-cadena-auto.cpp test-auto.cpp test-auto.hpp fecha.hpp cadena.hpp

OBJETOS = cadena.o fecha.o