SEMINARIO C++

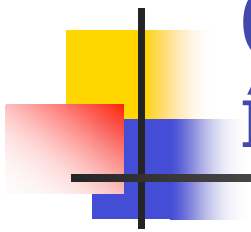
Introducción a la Programación Orientada a Objetos

Herramientas de programación y Práctica 0
v. 20070918

Cristina Cachero
Pedro J. Ponce de León

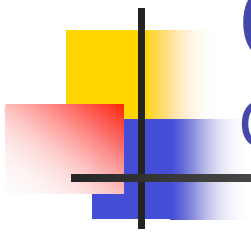
Depto. Lenguajes y Sistemas Informáticos - Universidad de Alicante

Universitat d'Alacant
Universidad de Alicante



C++
ÍNDICE

1. **Compilador: g++**
2. Directivas de compilación
3. Make
4. Doxygen
5. Utilidad de compresión tar
6. UML
7. Enunciado práctica 0



C++

COMPILADOR: G++

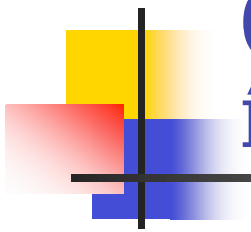
- GCC es una colección de compiladores para varios lenguajes: *C, C++, Objective C, Fortran y Java.* Incluye las librerías para éstos. Para C++ usaremos el compilador **g++.**

Sintaxis

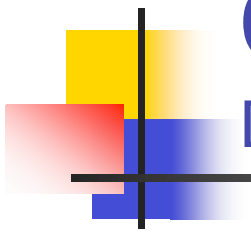
```
$ g++ [opciones] nom_fichero...
```

Opciones: con 'man gcc' podemos ver todas.

- **-help** → Indica a gcc que muestre su salida de ayuda
- **-o <fichero>** → El archivo ejecutable generado por gcc es por defecto `a.out`. Mediante este modificador, le especificamos el nombre del ejecutable.
- **-Wall** → No omite la detección de ningún warning. Por defecto, gcc omite una colección de warnings ``poco importantes''.
- **-c** → Preprocesa, compila y ensambla, pero no enlaza.
- **-g** → Incluye en el binario información necesaria para utilizar un depurador (**gdb** o **ddd**) posteriormente.

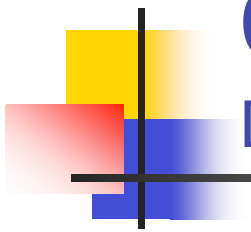


C++
ÍNDICE



C++
DIRECTIVAS DE COMPILACIÓN

- C++ ofrece la posibilidad de compilación condicional mediante la inclusión de ciertas directivas que controlan el comportamiento del preprocesador, de forma que este puede ignorar o compilar determinadas líneas del código en función de ciertas condiciones que son evaluadas durante el preproceso.



C++
DIRECTIVAS DE COMPILACIÓN

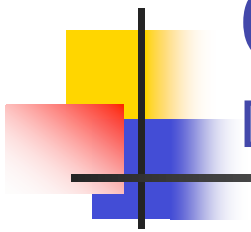
- **#ifdef…endif, #ifndef…#endif**
  - Son condicionales especializadas en comprobar si un macro-identificador está definido (con **#define**) o no.

Sintaxis

```
#ifdef identificador
#ifndef identificador
```

- Un macro-identificador X se define con **#define X** y se anula su definición con **#undef X**, con lo que podemos controlar a voluntad las zonas de código en que **X** se considera definido e indefinido.

- **Ejercicio**:
  - Añade a rectangulo.cpp un segundo #include "rectangulo.h". Intenta compilar el código de nuevo. ¿Qué ocurre?
  - Modifica los ficheros rectangulo.h, rectangulo.cpp y main.cpp con las directivas,#ifdef, #ifndef, #endif que consideres necesarias. Compila y comprueba que todo funciona correctamente.

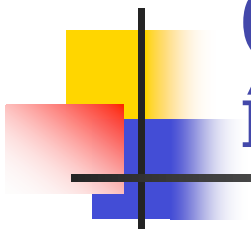


C++
DIRECTIVAS DE COMPILACIÓN

- La siguiente situación es muy habitual en proyectos en C++:
  - El fichero B.h incluye a A.h
  - El fichero C.h incluye a A.h
  - El fichero D.cpp incluye a B.h y a C.h

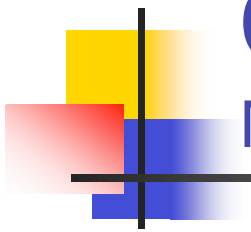
- Para evitar problemas al compilar por la doble inclusión de ficheros de declaraciones (.h) se inserta las siguientes directivas en cada .h:

```
#ifndef __A_H
#define __A_H
// declaraciones en el .h
#endif
<fin del fichero>
```



C++
ÍNDICE

1. Compilador: g++
2. Directivas de compilación
3. **Make**
4. Doxygen
5. Utilidad de compresión tar
6. UML
7. Enunciado práctica 0



C++
MAKE

- `make:` Utilidad que automatiza el proceso de compilación
- Busca un fichero de configuración `makefile` o `Makefile` (en ese orden)
- Ejemplos:
  - `make`
  - `make prog1 #crea el objetivo prog1`
  - `make -f mimakefile` (fich. config. mimakefile)
  - `make -f mimakefile prog1`

C++
El fichero de configuración `Makefile`

- Makefile de ejemplo que automatiza el comando gcc necesario para compilar el ejemplo del rectángulo.

```
$ make
$ make clean
```

```
OBJ = main.o rectangulo.o
OPC = -g -Wall
COMP = g++
.PHONY = clean
main: $(OBJ)
	$(COMP) $(OPC) $(OBJ) -o main
main.o: main.cpp rectangulo.h
	$(COMP) $(OPC) -c main.cpp
rectangulo.o: rectangulo.cpp rectangulo.h
	$(COMP) $(OPC) -c rectangulo.cpp
clean:
	rm main
	rm -r *.o
```

objetivo

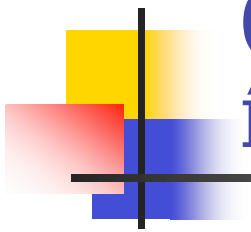
Objetivo ficticio

Órden de creación del ejecutable

El propio fichero y los que se incluyan en él

Órden de creación del *.o*

Tabuladores

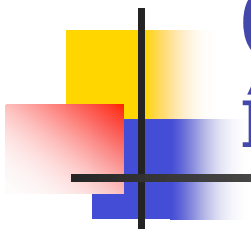


C++
ÍNDICE

1. Compilador: g++
2. Directivas de compilación
3. Make
4. **Doxygen**
5. Utilidad de compresión tar
6. UML
7. Enunciado práctica 0

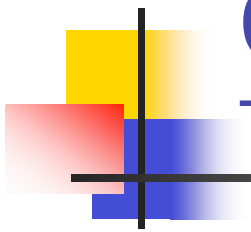
C++
DOXYGEN

- Herramienta para la generación de documentación (HTML entre otros formatos) a partir de comentarios en el código fuente.
- Descargad y consultad tutorial de Doxygen en la sección de materiales del C.V.



C++
ÍNDICE

1. Compilador: g++
2. Directivas de compilación
3. Make
4. Doxygen
5. **Utilidad de compresión *tar***
6. UML
7. Enunciado práctica 0



C++
TAR y MCOPY

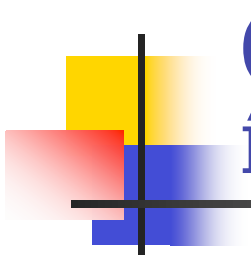
- Usar comando *tar* (comprimir) y *mcopy* para guardar los archivos en un disco:

```
$ tar cvzf prac1.tgz *
$ mcopy prac1.tgz a:/
```

  - Si quiero copiar texto en lugar de binario:
    - `$ mcopy -t Coordenada.cpp a:/`

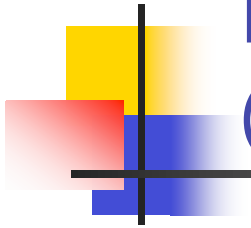
- Para recuperarlo del disco y descomprimirlo en la siguiente sesión:

```
$ mcopy a:/prac1.tgz
$ tar xvzf prac1.tgz
```



C++
ÍNDICE

1. Compilador: g++
2. Directivas de compilación
3. Make
4. Doxygen
5. Utilidad de compresión *tar*
6. **UML**
7. Enunciado práctica 0

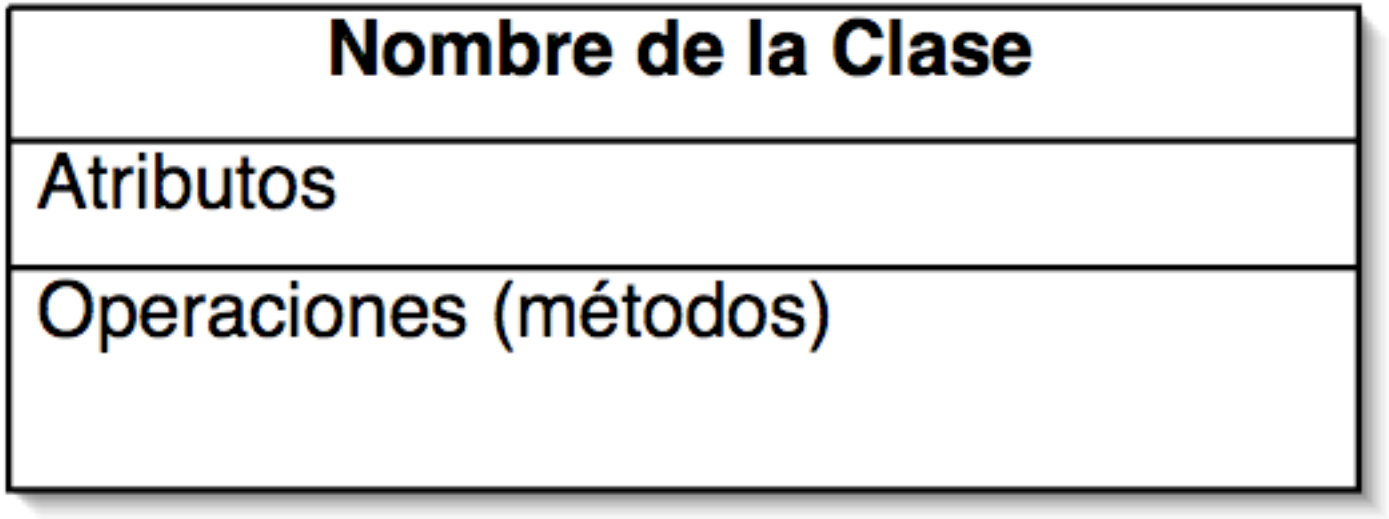


Breve introducción a UML
(Unified Modeling Language)

- UML es un lenguaje gráfico que se utiliza para el análisis y diseño de aplicaciones.
- Permite comunicar de forma precisa las ideas de diseño al equipo de desarrollo
- En POO usaremos únicamente el **diagrama de clases**
- Este diagrama especifica qué clases componen el sistema, sus atributos, su interfaz y sus relaciones con otras clases.
- En prácticas usaremos el diagrama de clases **detallado**

UML
Diagrama de clases

- Cada clase se representa por una caja con tres secciones:



UML
Diagrama de clases

Especificación de atributos y operaciones:

NombreClase
nombreAtributo : tipoAtributo [= valorPorDefecto]
nombreOperacion(<lista de argumentos>) : tipoDevuelto

- El valor por defecto de un atributo es opcional
- La lista de argumentos de un método se especifica del mismo modo que los atributos, separando los argumentos por comas. Se puede omitir el nombre de los argumentos e indicar únicamente su tipo.

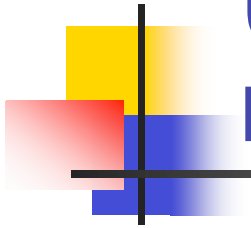
UML
Diagrama de clases

Visibilidad pública (+) y privada (-):

NombreClase
+ atributoPublico : tipoX - atributoPrivado : tipoY
+ metodoPublico(<lista de argumentos>) : tipoDevuelto - metodoPrivado(<lista de argumentos>) : tipoDevuelto

Métodos constantes:

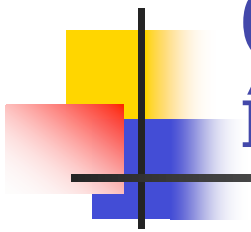
NombreClase
+ <<const>> metodoConstante(<lista de argumentos>) : tipoDevuelto



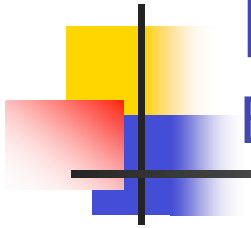
UML

Diagrama de clases

- La especificación de relaciones entre objetos y entre clases, así como otras características de los diagramas de clases se tratará en sesiones posteriores.



C++
ÍNDICE

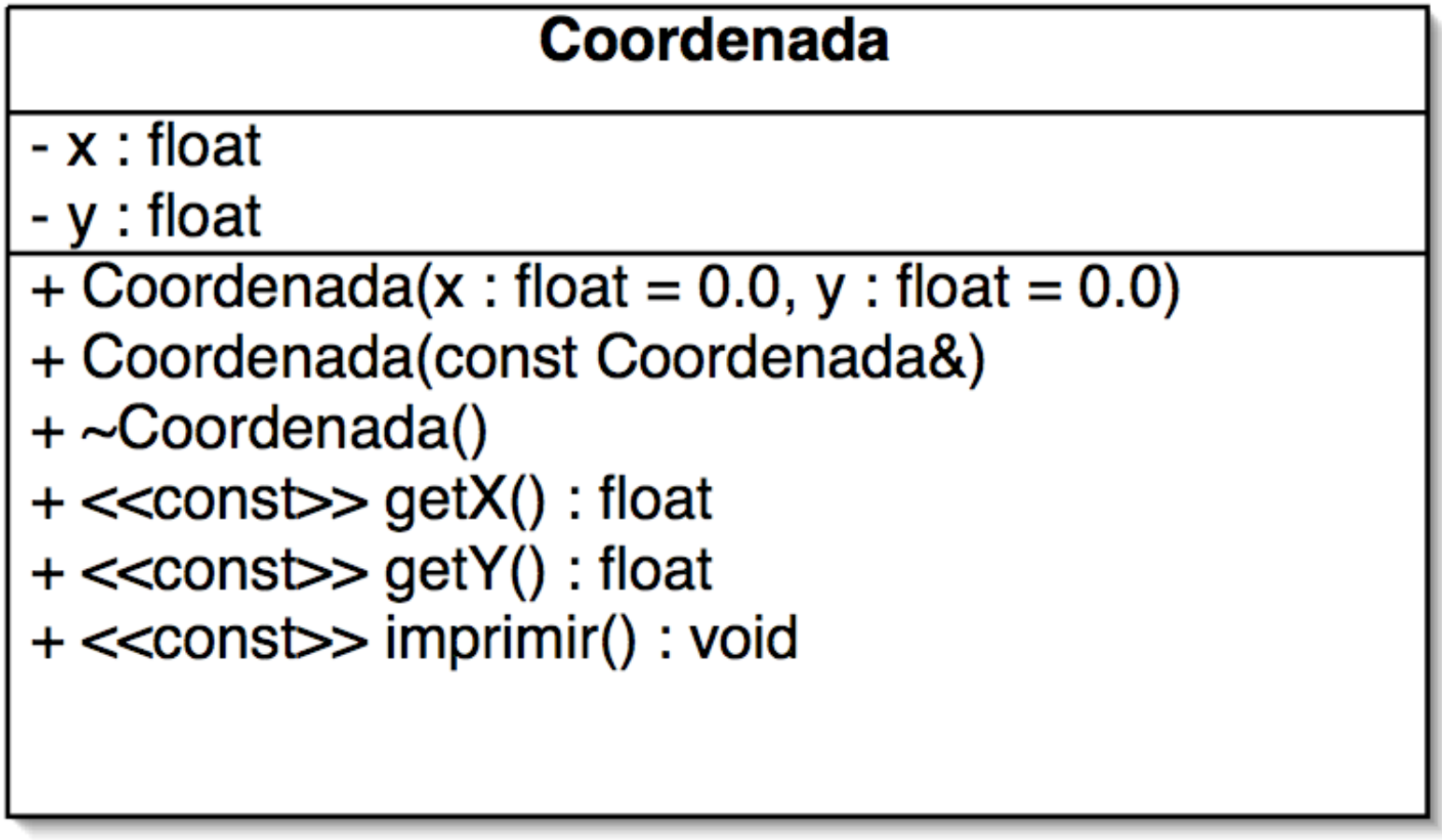


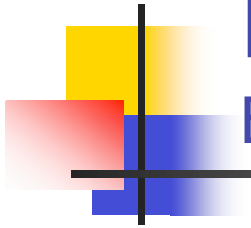
PRÁCTICA 0
ENUNCIADO

Los objetivos de esta pequeña práctica son:

- Introducirnos en la P.O.O.
- Implementar una clase sencilla diseñada en UML
- Familiarizarnos con la estructura de directorios del código fuente y el proceso de compilación.
- Realizar una entrega de prueba en el servidor de prácticas del DLSI

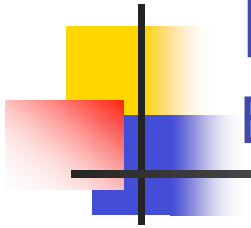
PRÁCTICA 0
ENUNCIADO





PRÁCTICA 0
ENUNCIADO

- Dado el diagrama de clases de la figura anterior, se pide:
  - Implementad la clase **Coordenada** mediante dos ficheros
    - *Coordenada.h*: declaración de métodos y variables. El fichero debe ubicarse en el directorio **include**
    - *Coordenada.cc*: implementación de métodos. Este fichero debe ubicarse en el directorio **lib**
  - Cread un fichero *main.cc* con una función `main()` donde se pidan al usuario un par de números con los que debéis crear una coordenada que debéis imprimir por pantalla. Escribid el código necesario para que cada método público de la clase sea utilizado al menos una vez. Este fichero debe ubicarse en el directorio **src**.
  - El método `Coordenada::imprimir()` debe imprimir los valores 'x' e 'y' de la coordenada separados por una coma.
- Comprimid toda la estructura de directorios en un fichero llamado **p0-0708.tgz**
  - `tar -czvf p0-0708.tgz *`

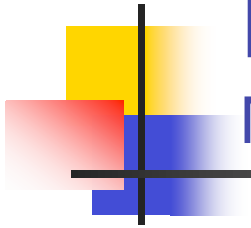


PRÁCTICA 0
EVALUACIÓN DE LA PRÁCTICA

- **Los requisitos imprescindibles para considerar correcta la práctica son:**
  - La práctica debe funcionar sin errores. En particular, no se debe producir ningún error del tipo "segmentation fault", "null pointer assignment", etc.
  - No se deben utilizar variables globales
  - La **entrada de datos debe estar filtrada.**
    - Los valores introducidos deben ser mayores o igual a cero
  - La práctica debe compilar correctamente con la orden **make** desde línea de comandos.

PRÁCTICA 0
FICHERO MAKEFILE

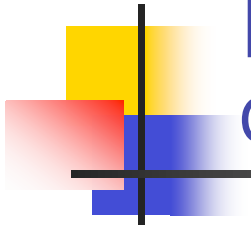
```
# Este fichero no se debe modificar. Lo puedes encontrar en la sección de materiales del CV.
# La practica debe compilar con este fichero situado en el directorio raíz del proyecto
# (del cual cuelgan lib/ src/ e include/).
#
COMP=g++
OPC=-g -Wall

.PHONY=clean

main: ./src/main.cc Coordenada.o ./include/Coordenada.h
    $(COMP) $(OPC) ./src/main.cc Coordenada.o -I ./include -o main

Coordenada.o: ./lib/Coordenada.cc ./include/Coordenada.h
    $(COMP) $(OPC) -c ./lib/Coordenada.cc -I ./include

clean:
    rm main
    rm -r *.o
```



PRÁCTICA 0

MODIFICACIÓN (opcional)

- **Cread un nuevo main() que genere dos números aleatorios, uno para la coordenada 'x' y otro para la coordenada 'y', ambos entre 0 y 100, los utilice para crear una coordenada y la muestre por pantalla.**



PRÁCTICA 0
GENERACION DE NUMEROS ALEATORIOS

- Funciones `rand()` y `srand()`
  - Permiten generar números (pseudo)aleatorios.
  - Los generadores de números aleatorios (Random Number Generator, RNG) se inicializan con un número llamado '**semilla'.**
  - Cada vez que se inicializa el RNG con la misma semilla, produce la misma secuencia de números aleatorios.
  - `srand()` inicializa el RNG.
  - `rand()` produce un entero aleatorio entre `0` y `RAND_MAX` (definido en `stdlib.h`).

PRÁCTICA 0
GENERACION DE NUMEROS ALEATORIOS

- Por cada ejecución de la práctica, se debe inicializar el RNG una sola vez. Esto se suele hacer en main() o, si estamos siguiendo una estructura puramente OO, en el método run() de la clase que representa a la aplicación:

```
#include <time.h> // para la funcion time()
#include <stdlib.h> // rand() y srand()
main() {
  srand((unsigned)time(NULL));
  …
}
```

- Después, cada vez que se necesite un entero aleatorio, se debe llamar a rand():
  - `int num = rand();`
- Si lo que quieres obtener es un entero entre 0 y N-1:
  - `num = (int)(N*(rand()/(RAND_MAX+1.0)))`