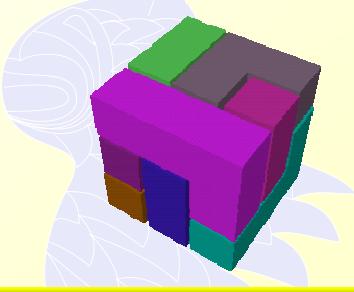


Metodología de la Programación

Compilación (*linux y g++*) y modularización

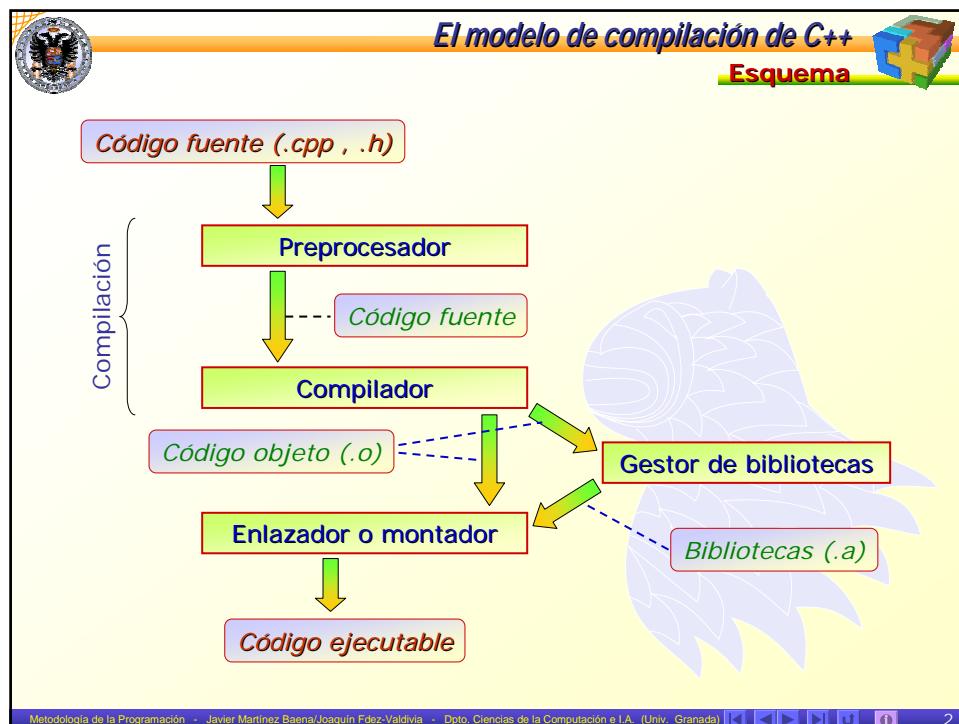
- El modelo de compilación de C++*
- El preprocesador de C++*
- El compilador g++*
- Construcción de bibliotecas*
- La modularización*



UGR Universidad de Granada

DECSAI Universidad de Granada

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons]



 **El modelo de compilación de C++** 

Las etapas principales

El preprocesador	<i>(Está integrado en el compilador: g++ -E)</i>
Recibe el código fuente (.h y .cpp)	
Devuelve el código fuente preprocesado:	
Elimina los comentarios	
Interpreta y procesa las directivas del preprocesador	
El compilador	<i>(g++ o gcc)</i>
Recibe el código fuente preprocesado	
Análisis léxico, sintáctico, semántico	Optimización
Devuelve el código objeto (casi código máquina)	
El enlazador (o montador o linker)	<i>(ld)</i>
Recibe código objeto y bibliotecas	<i>Alguno de los ficheros objeto debe tener una función main()</i>
Devuelve el ejecutable (código máquina)	

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  3

 **Metodología de la Programación** 

Compilación (linux y g++) y modularización

- El modelo de compilación de C++**
- El preprocesador de C++**
- El compilador g++**
- Construcción de bibliotecas**
- La modularización**



 **ugr** Universidad de Granada  **DECSAI** Universidad de Granada

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  4

El preprocesador de C++

La directiva #define

#define Permite definir constantes simbólicas

```
#define identificador texto_para_sustituir
```

Se hace una búsqueda y sustitución desde que aparece hasta el final del archivo (o hasta que haya un **#undef identificador**)

```
#define TAM 256
...
int v[TAM];
...
for (int i=0; i<TAM; i++) {
    ...
}
```

...

```
int v[256];
...
for (int i=0; i<256; i++) {
    ...
}
```



```
#define repetir for
...
repetir (int i=0; i<10; i++)
    ...

```

...

```
for (int i=0; i<10; i++)
    ...

```

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 5

El preprocesador de C++

La directiva #undefine

#undef Permite eliminar la definición de constantes simbólicas

```
#undef identificador
```

```
#define TAM 256
...
int v[TAM];
...
for (int i=0; i<TAM; i++)
    ...
...
#define TAM
...
for (int i=0; i<TAM; i++)
    ...

```

...

```
int v[256];
...
for (int i=0; i<256; i++)
    ...
...
for (int i=0; i<TAM; i++)
    ...

```

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 6

El preprocesador de C++

La directiva #define para crear macros

#define También se usa para crear macros

```
#define prototipo cuerpo
```

Conceptualmente son **similares** a funciones. Diferencias:

- No se hacen llamadas en tiempo de ejecución (+ eficiente, + tamaño)
- En general, las macros recursivas no funcionan
- No hay comprobación de tipos

```

#define doble(x) x+x
...
int y;
...
y=doble(2);
y=doble(3+h);
y=doble(3)*4;

```

```

...
int y;
...
y=2+2;
y=3+h+3+h;
y=3+3*4;

```

Error

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 7

El preprocesador de C++

La directiva #define para crear macros

#define doble(x) x+x → Tiene problemas

```

#define doble(x) (x+x)
...
y=doble(3)*4;

```

```

y=(3+3)*4;

```

```

#define cuad(x) (x*x)
...
y=cuad(3);
y=cuad(4+7);

```

```

y=(3*3);
y=(4+7*4+7);

```

Error

```

#define cuad(x) ((x)*(x))
...
y=cuad(4+7);

```

```

y=((4+7)*(4+7));

```

Vale para cualquier tipo !!!

```

#define MAX(x,y) ((x)>(y)?(x):(y))
...
integer y=MAX(4+7,5);
float z=MAX(7.2,6.56);

```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 8

El preprocesador de C++

La directiva #include

#include Permite incluir el contenido de un fichero

#include <fichero> Los ficheros son buscados en los directorios por defecto del sistema o en los indicados en la línea de órdenes (opción -I).

#include "fichero" Los ficheros se buscan en el directorio en donde se hace el #include. Y luego como en <>

Funciona como si hiciésemos un “copiar y pegar”.

```

fcab.h
#define T 120
int x=3;
void f(int y) {
    return T*y;
}

ejemplo.cpp
#include "fcab.h"
int main() {
    int x=f(T);
}

int x=3;
void f(int y) {
    return 120*y;
}
int main() {
    int x=f(120);
}
  
```

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 9

El preprocesador de C++

La directiva #include

carpeta

- fcab.h
- ejemplo1.cpp
- ejemplo2.cpp
- ejemplo3.cpp
- ejemplo4.cpp

fcab.h

...
Cuales son correctos?

ejemplo1.cpp

```
#include "iostream"
#include "fcab.h"
using namespace std;
...
```

ejemplo2.cpp

```
#include <iostream>
#include <fcab.h>
using namespace std;
...
```

ejemplo3.cpp

```
#include <iostream>
#include "fcab.h"
using namespace std;
...
```

ejemplo4.cpp

```
#include "iostream"
#include <fcab.h>
using namespace std;
...
```

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 10

El preprocesador de C++

La directiva #if

La directiva `#if` permite realizar la compilación condicional de ciertos trozos del programa.

```
#if <expresión>
    <bloque_si_verdad>
#else
    <bloque_si_falso>
#endif
```

```
#if <expresión1>
    <bloque_si_verdad_exp1>
#elif <expresión2>
    <bloque_si_verdad_exp2>
#elif <expresión3>
    <bloque_si_verdad_exp3>
...
#else
    <bloque_si_falsas_todas>
#endif
```

Las expresiones se evalúan en la etapa de preprocessado.

(antes de comenzar la ejecución)

```
#define TAM 128
#if TAM==120
#define X 200
int var=TAM;
#else
#define Y 300
float cad="Hola";
#endif
```

```
float cad="Hola";
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) | Back | Next | Previous | Home | 11

El preprocesador de C++

La directiva #if

Las expresiones de `#if` pueden incluir:

- Constantes enteras
- Constantes de tipo carácter (*cuidado: NO cadena*)
- Macros (`#define`)

Cualquier otro identificador se asume que es la constante cero

<pre>#define x 9 #if (x==9) #define OK #endif</pre>	Se define OK
<pre>int x=9 #if (x==0) #define OK #endif</pre>	Se define OK
<pre>int x=9 #if (x==9) #define OK #endif</pre>	NO se define OK
<pre>#define x 'a' #if (x=='a') #define OK #endif</pre>	Se define OK

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) | Back | Next | Previous | Home | 12

 **El preprocesador de C++** 

La directiva #if

Es frecuente usar la compilación condicional para hacer código portable entre distintos sistemas:

```
#define SISTEMA 'L'

#if SISTEMA=='L'
    #define CABECERA "linux.h"
#elif SISTEMA=='W'
    #define CABECERA "windows.h"
#elif SISTEMA=='M'
    #define CABECERA "macos.h"
#endif

#include CABECERA
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  13

 **El preprocesador de C++** 

Las directivas #ifdef y #ifndef

Esta vez, el objetivo es realizar la compilación de un trozo de programa condicionada a que haya sido definida o no una constante simbólica (`#define`).

```
#ifdef identificador
...
#endif

#ifndef identificador
...
#endif
```

```
#if SISTEMA=='L'
    #define CABECERA "linux.h"
#elif SISTEMA=='W'
    #define CABECERA "windows.h"
#elif SISTEMA=='M'
    #define CABECERA "macos.h"
#endif

#ifdef CABECERA
    #include CABECERA
#else
    #include "otro.h"
#endif
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  14

 **El preprocesador de C++** 

La directiva #if

Lo más habitual en estos casos:

```
#define LINUX

#ifndef LINUX
    #include "linux.h"
#endif

#ifndef WINDOWS
    #include "windows.h"
#endif

#ifndef MCINTOSH
    #include "macos.h"
#endif
```

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  15

 **Metodología de la Programación** 

Compilación (*linux* y *g++*) y modularización

- El modelo de compilación de C++*
- El preprocesador de C++*
- El compilador g++*
- Construcción de bibliotecas*
- La modularización*

 **ugr** Universidad de Granada  **DECSAI** Universidad de Granada

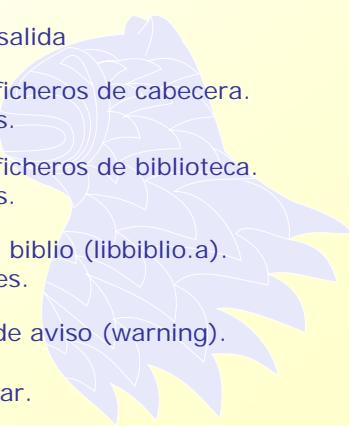
Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  16

El compilador g++

Las opciones de compilación

g++ opciones ficheros_de_entrada

- c** Sólo preprocesa y compila (no enlaza).
El resultado es el código objeto (.o)
- o fichero** Nombre del fichero de salida
- Ipath** Donde se deben buscar los ficheros de cabecera.
Se puede poner varias veces.
- Lpath** Donde se deben buscar los ficheros de biblioteca.
Se puede poner varias veces.
- lbiblio** Usar la biblioteca llamada biblio (libbiblio.a).
Se puede usar varias veces.
- Wall** Mostrar todos los mensajes de aviso (warning).
- g** Genera información para depurar.



Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) | Back | Forward | Home | Help | 17

El compilador g++

Ejemplo de uso

fcab

- cabecera1.h
- cabecera2.h
- codfuen
- codigol.cpp
- codigo2.cpp
- main.cpp
- blibs
- libmibib.a

(contiene #include "cabecera1.h")

(contiene #include "cabecera2.h")

Contiene #include "cabecera1.h"

Hace uso de funciones de la biblioteca llamada mibib (libmibib.a)

Los #include se buscan en "../fcab"

g++ -c codigol.cpp -I../fcab -o codigol.o

Sólo compilar Fichero de entrada El resultado de compilar lo pones en "codigol.o"

g++ -c codigo2.cpp -I../fcab -o codigo2.o

g++ -c main.cpp -I../fcab -o main.o

g++ main.o codigol.o codigo2.o -L../blibs -lmibib -o main

Enlaza con libmibib.a

Enlazar (no hay -c) Las bibliotecas se buscan en "../blibs"

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) | Back | Forward | Home | Help | 18

El compilador g++

Una opción interesante ...

g++ -E fichero_de_entrada

Con esta opción podemos ver el resultado del preprocesador.

cab1.h	cab2.h	main.cpp
<pre>#ifndef __CAB1_H__ #define __CAB1_H__ #include "cab2.h" typedef char caracter; const int cte1 = 3; // Comentario de cab1.h double func1(int x); #define CTE 44 const int cte3 = CTE; #endif</pre>	<pre>#ifndef __CAB2_H__ #define __CAB2_H__ enum Dia {Lunes,Martes,Miercoles, Jueves,Viernes,Sabado, Domingo}; const int cte2 = 6; // Comentario de cab2.h float func2(int x, Dia y); #endif</pre>	<pre>int z=3; #include "cab1.h" // Esta es una aplicación para // probar el uso del // preprocesador int main(int argc,char *argv[]) { cout<<"Ejemplo de uso de -E" << endl; /* Observa que no hemos hecho el #include deiostream y, por lo tanto, se producirán errores de compilación. */ }</pre>

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 19

El compilador g++

Una opción interesante ... g++ -E main.cpp

<pre># 1 "main.cpp" # 1 "<built-in>" # 1 "<command line>" # 1 "main.cpp" 1 int z = 3; # 1 "cab1.h" 1 2 3 4 5 # 1 "cab2.h" 1 1 2 3 4 5 enum Dia {Lunes, Martes, Miércoles, Jueves, Viernes, Sabado, Domingo}; 6 const int cte2 = 6; 7 8 float func2(int x, Dia y); # 6 "cab1.h" 2 6 typedef char caracter; 7 const int cte1 = 3; 8 9 double func1(int x);</pre>	<p>Comienza la lectura de main.cpp</p> <p>Comienza la lectura de cab1.h</p> <p>Procesa #ifndef __CAB1_H__</p> <p>Procesa #define __CAB1_H__</p> <p>Comienza la lectura de cab2.h</p> <p>Procesa #ifndef __CAB2_H__</p> <p>Procesa #define __CAB2_H__</p> <p>Elimina comentario</p> <p>Regresa a cab1.h (línea 6)</p> <p>Elimina comentario</p>
---	--

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 20

El compilador g++

Una opción interesante ... g++ -E main.cpp

```

6      Jueves, Viernes, Sabado,
7      Domingo};
8  const int cte2 = 6;
9
10 float func2(int x, Dia y);
# 6 "cab1.h" 2
6  typedef char caracter;
7  const int cte1 = 3;
8
9  double func1(int x);
10
11
12 const int cte3 = 44;
# 4 "main.cpp" 2
4
5
6
7
8  int main(int argc, char *argv[])
9  {
10     cout << "Ejemplo de uso de -E"
11         << 44 << endl;
12
13
14
15
16 }

```

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 21

Metodología de la Programación

Compilación (linux y g++) y modularización

- El modelo de compilación de C++*
- El preprocesador de C++*
- El compilador g++*
- Construcción de bibliotecas*
- La modularización*

ugr Universidad de Granada

DECSAI
Universidad de Granada

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 22

Construcción de bibliotecas

¿Qué son? ¿Para qué sirven? ¿Cómo se construyen?

```

graph TD
    obj1[objeto1.o] --> cmd[ar rsv libmibib.a]
    obj2[objeto2.o] --> cmd
    obj3[objeto3.o] --> cmd
    cmd --> lib[libmibib.a]
  
```

Permiten reutilizar código.

Biblioteca (*library*) = conjunto de ficheros objeto con relación entre sí.

ar opciones biblioteca ficheros_objeto

rsv
r = Añadir reemplazando
s = Crear índice
v = "Verbose"

Metodología de la Programación

Compilación (*linux y g++*) y modularización

- El modelo de compilación de C++**
- El preprocesador de C++**
- El compilador g++**
- Construcción de bibliotecas**
- La modularización**

ugr Universidad de Granada

DECSAI
Universidad de Granada

 **La modularización** 
¿Qué es?

Modularizar → Compartimentar
 Problema grande → Descomposición en problemas pequeños (módulos)

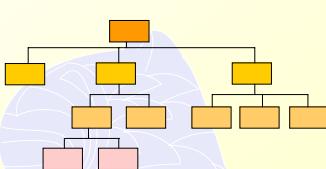
Modularizar permite:	Trabajo en equipo Facilita depuración (aisla errores) Mejorar legibilidad y modificabilidad Elimina redundancias de código Reusabilidad de código
Conceptos importantes:	Abstracción (funcional, de datos) Encapsulamiento / Ocultamiento de información Acoplamiento Cohesión Diseño descendente (top-down) Compilación separada

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  25

 **La modularización** 
¿Cómo se modulariza un programa?

División en módulos pequeños → **Diseño descendente (top-down)**

- Primero pensamos en abstracto
- Realizamos sucesivas descomposiciones hasta que sea necesario

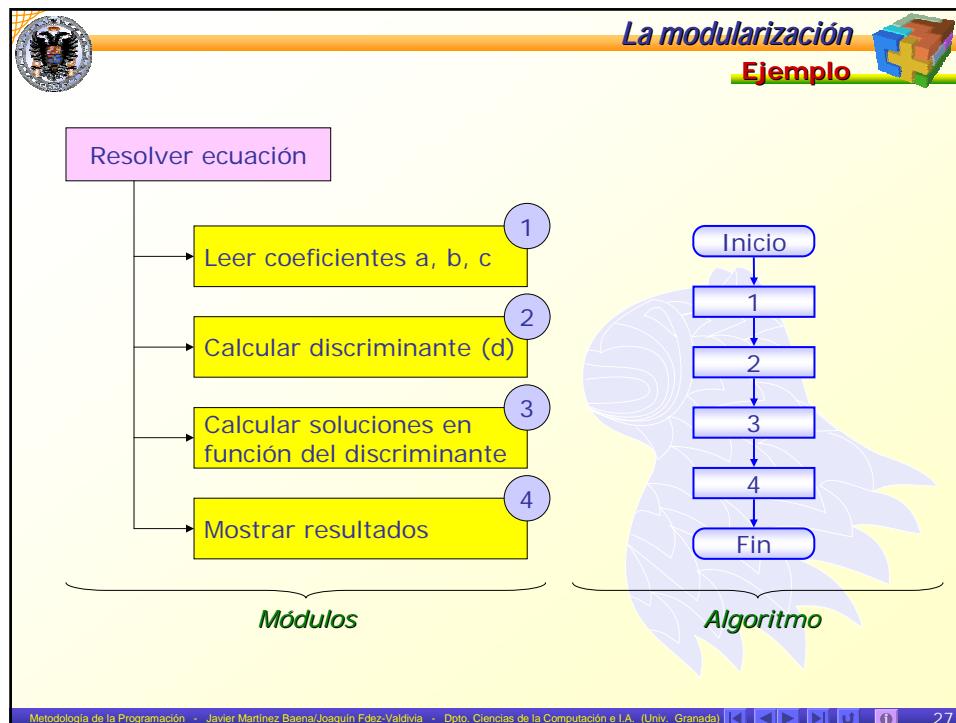
Cada módulo realiza una tarea única. Módulos pequeños. Módulos independientes.	
--	--

Ejemplo: Calcular solución de ecuación de 2º grado:

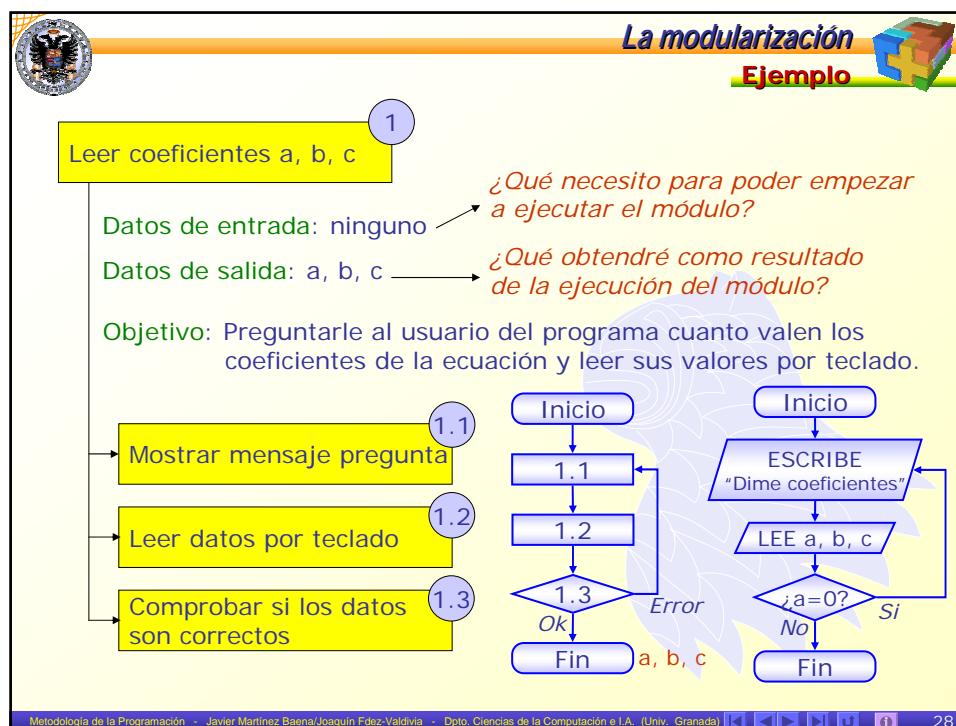
$ax^2 + bx + c = 0$	$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
---------------------	--

*Dependiendo del valor del discriminante ($b^2 - 4ac$), sabremos cuantas soluciones hay.
 Resolvemos sistemas con $a < 0$.*

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  26



Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 27



Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 28

La modularización

Ejemplo

2

Calcular discriminante (d)

Datos de entrada: salida del módulo 1 (a, b, c)
 Datos de salida: d (discriminante, número real)

No tiene sub-módulos

```

    graph TD
      Inicio([Inicio]) --> D[d = b2 - 4ac]
      D --> Fin([Fin])
      style D fill:#000,color:#fff
      style Fin fill:#000,color:#fff
  
```

¿Es suficiente el nivel de detalle?

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 29

La modularización

Ejemplo

3

Calcular soluciones en función del discriminante

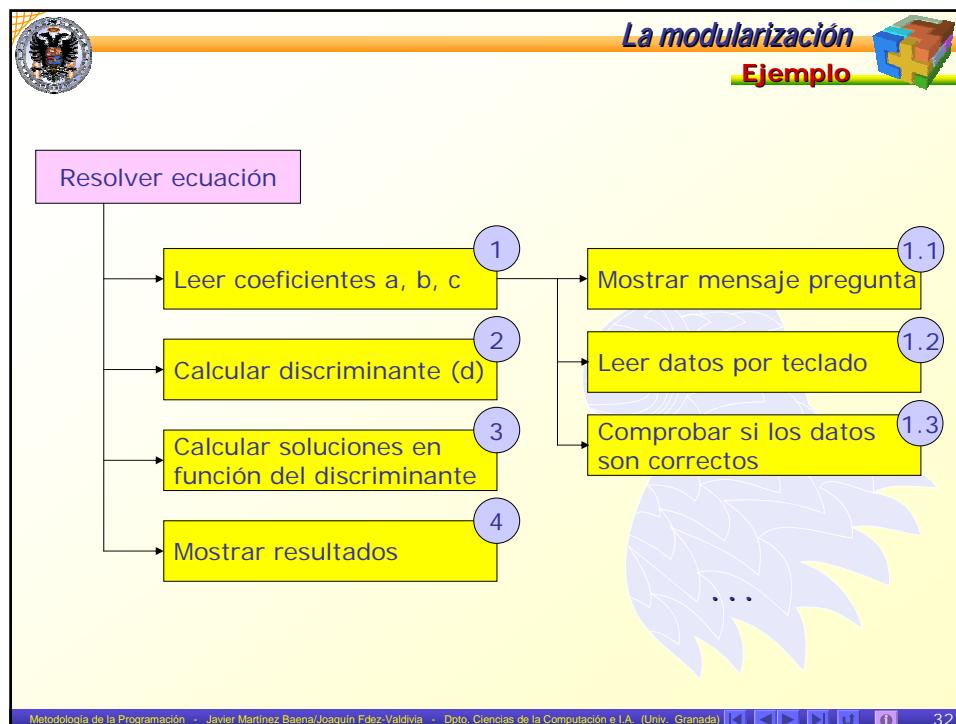
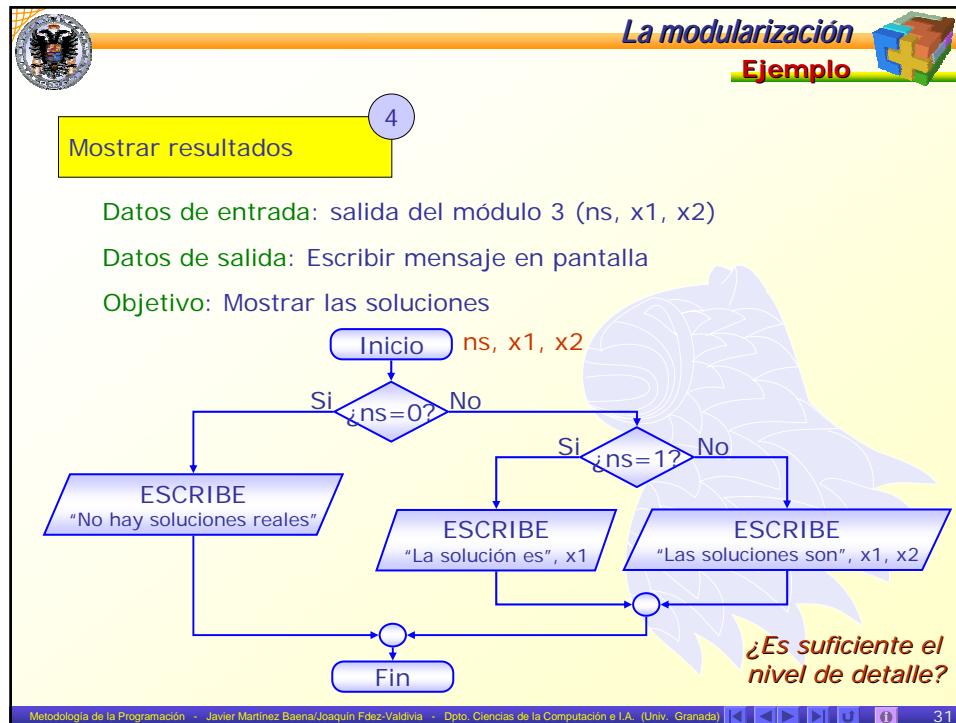
Datos de entrada: a, b, c, salida del módulo 2 (d) → Número entero
 Datos de salida: ns (*número de soluciones*), x1 (*1^a sol*), x2 (*2^a sol*) → Números reales
 Objetivo: Calcular las soluciones

```

    graph TD
      Inicio([Inicio]) --> D{Si ; d=0?}
      D -- No --> D2{Si ; d>0?}
      D -- No --> Fin([Fin])
      D --> ns1[ns = 1  
x1 = -b/(2a)]
      D2 -- Si --> ns2[ns = 2  
x1 = (-b+RAIZ(d))/(2a)  
x2 = (-b-RAIZ(d))/(2a)]
      D2 -- No --> ns0[ns = 0]
      ns1 --> Fin
      ns2 --> Fin
      ns0 --> Fin
  
```

¿Es suficiente el nivel de detalle?

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 30



 **La modularización** 

Características de los módulos

Los módulos (funciones) deben ser **pequeños** (pocas líneas de código).

Acoplamiento mínimo

- Se refiere a la independencia entre módulos.
- El funcionamiento interno de un módulo ha de ser lo más independiente posible del funcionamiento interno de otros módulos.
- *Ejemplo: las variables globales implican acoplamiento.*

Cohesión máxima

- Se refiere al grado de interrelación entre las partes un mismo módulo.
- Un módulo debe hacer tareas muy específicas y acotadas.
- *Ejemplo: un módulo que hace varias tareas tiene poca cohesión. Calcular a la vez la media y la moda de un vector.*

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  33

 **La modularización** 

Módulos a distintos niveles

Un módulo puede ser (en función del nivel al que nos movamos):

- Una función.
- Un fichero (que contiene varias funciones y datos).
- Un TDA (Tipo de Dato Abstracto).
- Una biblioteca (conjunto de ficheros y/o funciones y/o datos).
- Un namespace (agrupamiento lógico de funciones y datos).

Cuando se construyen programas grandes, las funciones y estructuras de datos se dividen y agrupan, según su uso, en múltiples ficheros y bibliotecas.

En esta situación, cada módulo suele tener:

- Interfaz de acceso al módulo (parte pública)
- Implementación del módulo (parte privada).

La división en ficheros facilita:

- Compilación separada
- Construcción y Mantenibilidad
- Trabajo en equipo

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  34

 **La modularización** 

Partes de un módulo

Interfaz (Parte pública)	Debe ser conocida por cualquiera que desee usar el módulo.
Se escribe en los ficheros de cabecera (.h) Suele contener: <code>#define</code> públicos Prototipos de funciones Definición de tipos públicos (<code>struct</code> , <code>class</code> , <code>typedef</code> , <code>enum</code> , ...)	
Implementación (Parte privada)	Sólo conocida por el programador o diseñador del módulo.
Se escribe en los ficheros de implementación (.cpp) Suele contener: La implementación de lo que se ha declarado en la parte pública e implementaciones privadas. <i>Al finalizar la construcción del módulo y compilarlo debemos generar, o bien un fichero objeto, o bien una biblioteca (esto lo decide el diseñador).</i>	

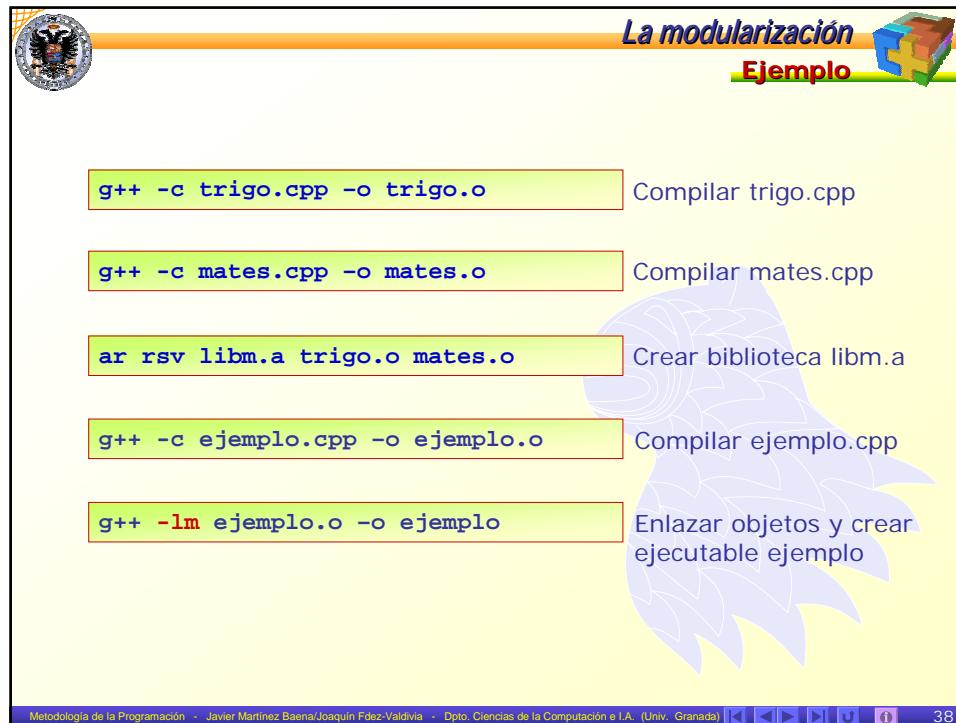
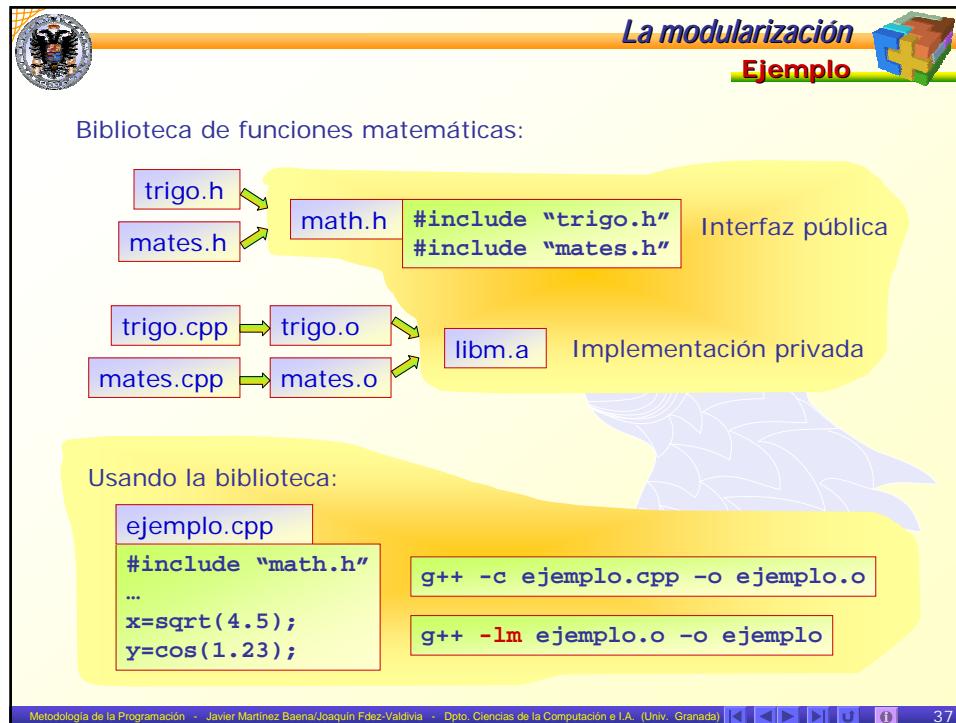
Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  35

 **La modularización** 

Ejemplo

Módulo de funciones trigonométricas		
trigo.h	<code>double sin(double x); double cos(double x); double tan(double x);</code>	<i>Interfaz pública</i>
trigo.cpp	Algoritmos que calculan las funciones	<i>Implementación privada</i>
Módulo de otras funciones matemáticas		
mates.h	<code>#define PI 3.14159 double sqrt(double x); double exp(double x); double log(double x);</code>	<i>Interfaz pública</i>
mates.cpp	Algoritmos que calculan las funciones	<i>Implementación privada</i>

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada)  36



La modularización

Compilación separada

```

graph TD
    trigo_cpp[trigo.cpp] -- "#include" --> trigo_h[trigo.h]
    mates_cpp[mates.cpp] -- "#include" --> mates_h[mates.h]
    mates_h -- "#include" --> ejem_cpp[ejemplo.cpp]
    ejem_cpp -- "#include" --> math_h[math.h]
    trigo_h -- "#include" --> math_h
    trigo_cpp -- "compilar" --> trigo_o[trigo.o]
    mates_cpp -- "compilar" --> mates_o[mates.o]
    mates_o -- "agrupar" --> libm_a[libm.a]
    trigo_o -- "agrupar" --> libm_a
    libm_a -- "enlazar" --> ejemplo_o[ejemplo.o]
    ejem_cpp -- "compilar" --> ejemplo_o
    ejemplo_o -- "enlazar" --> ejemplo[ejemplo]
  
```

El modelo de compilación separada sirve para acelerar la creación de proyectos grandes.

Si se modifica un módulo → NO hay que volver a compilar el proyecto completo. Sólo se compilan las partes afectadas por la modificación.

Dependencias en el ejemplo anterior:

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 39

La modularización

Compilación separada

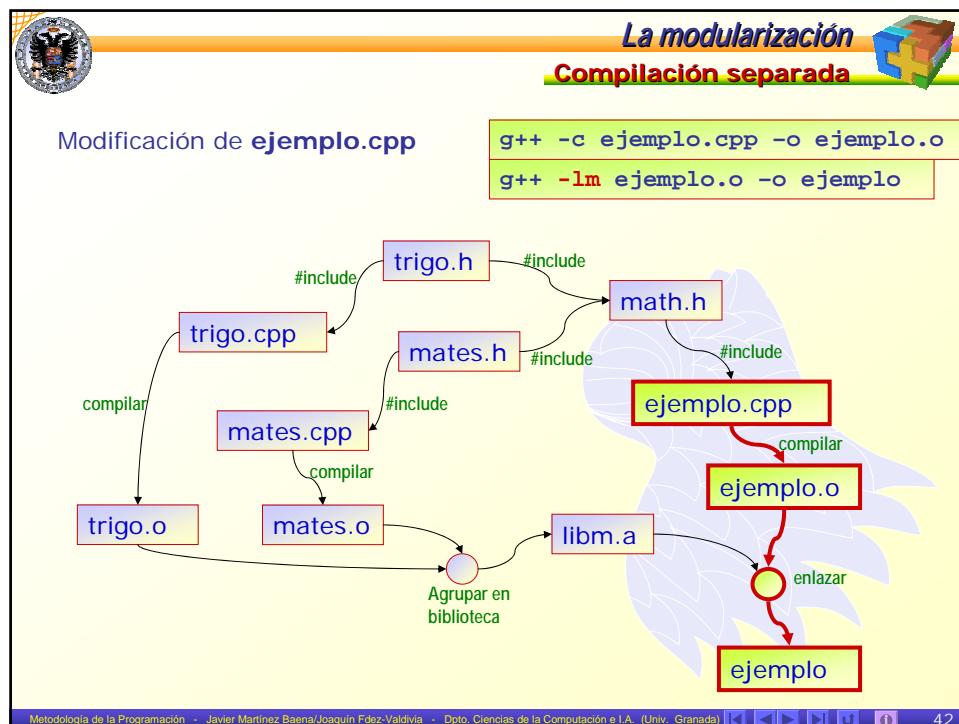
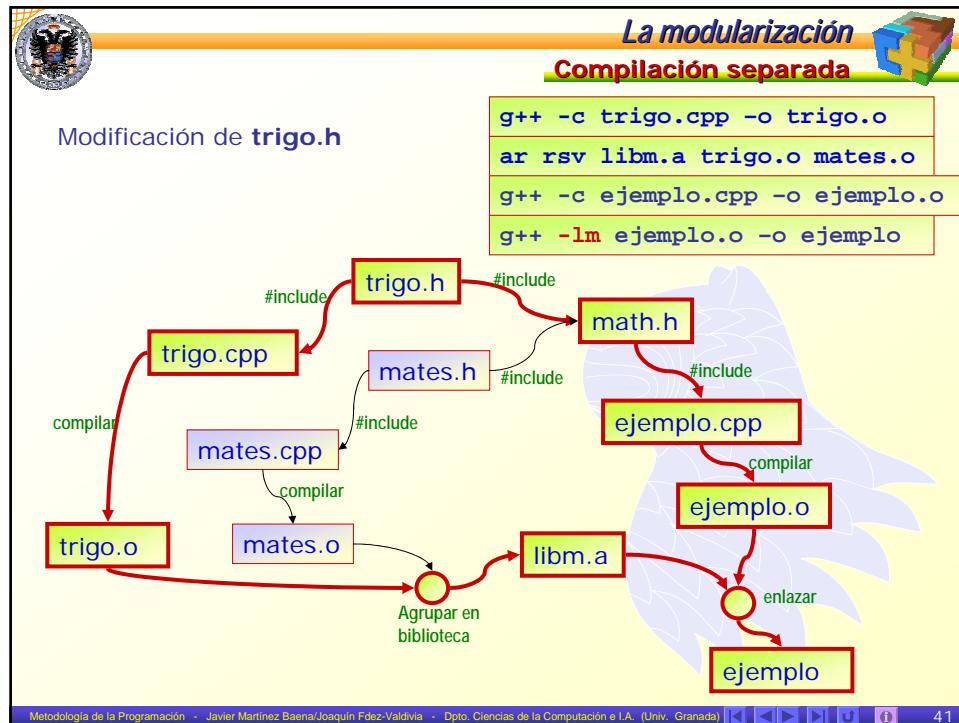
Modificación de `trigo.cpp`

```

graph TD
    trigo_cpp[trigo.cpp] -- "#include" --> trigo_h[trigo.h]
    mates_cpp[mates.cpp] -- "#include" --> mates_h[mates.h]
    mates_h -- "#include" --> ejem_cpp[ejemplo.cpp]
    ejem_cpp -- "#include" --> math_h[math.h]
    trigo_h -- "#include" --> math_h
    trigo_cpp -- "compilar" --> trigo_o[trigo.o]
    mates_cpp -- "compilar" --> mates_o[mates.o]
    mates_o -- "Agrupar en biblioteca" --> libm_a[libm.a]
    trigo_o -- "Agrupar en biblioteca" --> libm_a
    libm_a -- "enlazar" --> ejemplo_o[ejemplo.o]
    ejem_cpp -- "compilar" --> ejemplo_o
    ejemplo_o -- "enlazar" --> ejemplo[ejemplo]
  
```

`g++ -c trigo.cpp -o trigo.o`
`ar rsv libm.a trigo.o mates.o`
`g++ -lm ejemplo.o -o ejemplo`

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 40



La modularización

Construcción de los ficheros de cabecera (.h)

```

    mod1.h
    mod2.h
    #include "mod1.h"
    #include "mod2.h"

    ejemplo.cpp
    #include "mod1.h"
    #include "mod2.h"
  
```

Al compilar ejemplo.cpp:
 Se incluye mod1.h
 Se incluye mod2.h
 Se incluye mod1.h

Se incluye dos veces !!! Podría haber problemas

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 43

La modularización

Construcción de los ficheros de cabecera (.h)

```

    mod1.h
    #include "mod2.h"

    mod2.h
    #include "mod1.h"

    ejemplo.cpp
    #include "mod1.h"
  
```

Al compilar ejemplo.cpp:
 Se incluye mod1.h
 Se incluye mod2.h
 Se incluye mod1.h
 Se incluye mod2.h

...

Error: Se incluyen mutuamente de forma indefinida !!!

Metodología de la Programación - Javier Martínez Baena/Joaquín Fdez-Valdivia - Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [navigation icons] 44

La modularización

Construcción de los ficheros de cabecera (.h)

```

mod.h
#ifndef __MOD__H__
#define __MOD__H__
...
... contenido de mod.h
#endif

mod1.h
#ifndef __MOD1__H__
#define __MOD1__H__
#include "mod2.h"
...
#endif

mod2.h
#ifndef __MOD2__H__
#define __MOD2__H__
#include "mod1.h"
...
#endif

ejemplo.cpp
#include "mod1.h"

```

Identificador único

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 45

La modularización

Espacios de nombres (namespace)

Permiten **encapsular** trozos de código dentro de un nombre común.

Ejemplo: **std**

<i>Declaración</i> <pre>namespace identificador { ... int x=3; ... int func(int x) { ... } ... }</pre>	<i>Uso</i> <pre>... identificador::x = 4; z = identificador::func(2); ...</pre>
<i>Declaración</i> <pre>using namespace identificador;</pre>	<i>Uso</i> <pre>... x = 4; z = func(2); ...</pre>

Metodología de la Programación · Javier Martínez Baena/Joaquín Fdez-Valdivia · Dpto. Ciencias de la Computación e I.A. (Univ. Granada) [Navigation icons] 46