

“Práctica 0”

Repasa cómo compilar y enlazar,
uso de Makefiles y librerías

¿Qué *** es GCC?

- GCC = GNU Compiler Collection.
- Herramientas libres para el compilado y enlazado de código fuente.
- Soportan multitud de lenguajes de programación.
- Además son multi-plataforma (corren en Linux, Mac, Windows, etc).
- En este caso se utiliza GCC en GNU/Linux para compilar código C.

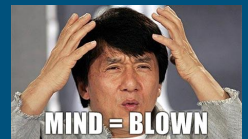
El primer paso es verificar que está instalado. Se ejecuta en la consola:

```
gcc --version
```

```
gcc (Ubuntu 5.2.1-22ubuntu2) 5.2.1 20151010  
Copyright (C) 2015 Free Software Foundation, Inc.  
This is free software; see the source for copying conditions. There is NO  
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

(la versión no tiene por qué coincidir)

El compilador GCC se utiliza para compilar las nuevas versiones del compilador GCC.



Compilando un hola mundo

En una carpeta “pruebas” crear el fichero “hola.c” con el siguiente contenido:

```
int main() {  
    print("ola k ase\n");  
    return 0;  
}
```

Desde consola, compilar con:

```
gcc -o hola_ejecutable hola.c
```

Esto genera un ejecutable que se puede invocar con:

```
./hola_ejecutable
```

Compilando un hola mundo



El compilador GCC avisa cuando hay problemas, y como es bastante listo muchas veces los resuelve por su cuenta:

```
$ mkdir pruebas
$ cd pruebas/
$ gcc -o hola_ejecutable hola.c
hola.c: In function 'main':
hola.c:2:5: warning: implicit declaration of function 'printf' [-Wimplicit-function-declaration]
      printf("ola k ase\n");
      ^
hola.c:2:5: warning: incompatible implicit declaration of built-in function 'printf'
hola.c:2:5: note: include '<stdio.h>' or provide a declaration of 'printf'
$ ./hola_ejecutable
ola k ase
$
```

- Pero ignorar los avisos de GCC es malo para la salud.

GCC está diciendo que no conoce la definición de “printf” y que se debe incluir el catálogo estándar “stdio.h”, o directamente la cabecera que define “printf”:

```
int printf ( const char * format, ... );
```

Otro ejemplo, usando la librería matemática

Si se incluye el catálogo `stdio.h` mediante el `#include` correspondiente al inicio del código, el error desaparece ya que GCC ahora sí encuentra la definición de `printf`.

```
$ gcc -o hola_ejecutable hola.c
$ ./hola_ejecutable
ola k ase
$
```



Ahora se modifica el código para realizar una simple operación matemática:

```
#include <stdio.h>
int main() {
    double valor = 16;
    double raiz = sqrt(valor);
    printf("La raiz cuadrada de %lf es %lf\n", valor, raiz);
    return 0;
}
```

Este código se puede compilar y enlazar simultáneamente con:

```
gcc -o hola_ejecutable hola.c
```

Otro ejemplo, usando la librería matemática

Tras ejecutar el comando, aparece el mismo problema de antes, pero ahora ni siquiera se genera un ejecutable:

```
$ gcc -o hola_ejecutable hola.c
hola.c: In function 'main':
hola.c:4:19: warning: implicit declaration of function 'sqrt' [-Wimplicit-function-declaration]
    double raiz = sqrt(valor);
                   ^
hola.c:4:19: warning: incompatible implicit declaration of built-in function 'sqrt'
hola.c:4:19: note: include '<math.h>' or provide a declaration of 'sqrt'
/tmp/cc0XWDVy.o: In function 'main':
hola.c:(.text+0x23): undefined reference to `sqrt'
collect2: error: ld returned 1 exit status
$ ./hola_ejecutable
bash: ./hola_ejecutable: No such file or directory
$
```



¿Bastará con incorporar la definición de sqrt()?

```
double sqrt(double x);
```

¿O quizás el catálogo matemático completo?

```
#include <math.h>
```

...¿consultar en stackoverflow? ¿en serio? →→

Otro ejemplo, usando la librería matemática

Tanto si se añade la definición de `sqrt()`, como si se incorpora el `#include <math.h>`, GCC ya no es capaz de solucionar el problema:

```
#include <stdio.h>
double sqrt(double x);
int main() {
    double valor = 16;
    double raiz = sqrt(valor);
    printf("La raiz cuadrada de %lf es %lf\n", valor, raiz);
    return 0;
}
```

```
$ gcc -o hola_ejecutable hola.c
/tmp/ccKVLG4l.o: In function 'main':
hola.c:(.text+0x23): undefined reference to 'sqrt'
collect2: error: ld returned 1 exit status
$ ./hola_ejecutable
bash: ./hola_ejecutable: No such file or directory
$
```

El código compila, pero falla el enlazado (se queja la herramienta ld).

- Es necesario indicarle a GCC dónde buscar el código de la función `sqrt()`. De este modo podrá enlazar juntos el binario del hola mundo *-compilado por GCC en un .o temporal-* y el código binario de la librería matemática.

Enlazado de librerías de sistema

Se ha visto que al utilizar la opción `-o` con el comando GCC es posible compilar y enlazar simultáneamente. Pues también es posible incorporar librerías manualmente al proceso de enlazado, para ello se utiliza la bandera `-l` (“menos L”) seguido del identificador de la librería deseada.

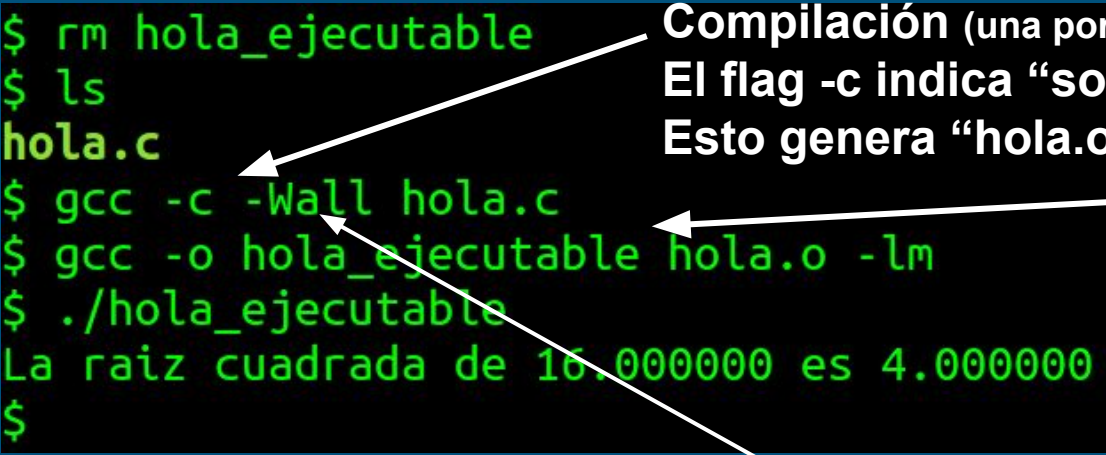
Por ejemplo, para incluir la librería matemática se utiliza “`-lm`”; para la librería de hilos POSIX se utiliza “`-lpthread`”; para las librerías de PulseAudio “`-lpulse`” y “`-lpulse-simple`”; para las de REDES II se utilizan `-lircinterface`, `-lircredes`, `-lirctad` o `-lsoundredes`. Y así con cualquier librería que se quiera usar y esté instalada en el sistema.

Con esto ya es posible compilar con éxito el ejemplo de antes:

```
$ gcc -o hola_ejecutable hola.c -lmath 🤪  
$ ./hola_ejecutable  
La raíz cuadrada de 16.000000 es 4.000000  
$
```


Separando compilación y enlazado

En proyectos con varios ficheros fuente y que usen librerías, lo normal es compilar y enlazar por separado. Por ejemplo:



A terminal window with a black background and green text. The commands and output are as follows:

```
* $ rm hola_ejecutable
$ ls
hola.c
$ gcc -c -Wall hola.c
$ gcc -o hola_ejecutable hola.o -lm
$ ./hola_ejecutable
La raíz cuadrada de 16.000000 es 4.000000
$
```

Compilación (una por cada fichero fuente)
El flag **-c** indica “solamente compilar”.
Esto genera “hola.o” (código binario).

Enlazado (uno por ejecutable)
El flag **-o** indica “enlazar”.
Genera un ejecutable que incorpora todos lo .o con los binarios necesarios de cada librería.

Arrows from the text annotations point to the corresponding flags in the terminal output: one arrow points from 'Compilación' to '-c', and another points from 'Enlazado' to '-o'.

*Se recomienda incorporar siempre el flag **-Wall** a la línea de compilación, que simplemente solicita a GCC que proporcione avisos más estrictos. En definitiva, fomenta el uso de buenas prácticas en el código.*

*nota: `rm` = eliminar. `ls` = listar directorio.

Más conceptos a revisar

- Los catálogos (“cabeceras”, “headers”, .h) no son lo mismo que las librerías (“bibliotecas”, .a, .so).
- *Por qué/Para qué/Cómo* se utilizan los archivos de cabecera:
<https://www.youtube.com/watch?v=n6BEuft6Fq4>
- Diferencia entre librerías estáticas (.a) y dinámicas (.so). Uso del comando “ar” para la creación de librerías estáticas propias.
<https://www.luzem.com/2009/10/18/librerias-estaticas-y-librerias-dinamicas/>



Automatizando el proceso con Makefiles

¡Tan cómodos que parecen magia!



El Makefile mas sencillo

En la carpeta “pruebas”, del último ejemplo, crear un fichero “Makefile” con el siguiente contenido:

```
all:
```

```
    gcc -o hola_ejecutable hola.c -lm
```



¡Ojo! Esto no son varios espacios, es un tabulador.

Es el mismo comando de antes, simplemente se le ha incorporado la etiqueta “all” para que sea la regla por defecto. De este modo basta con ejecutar “make”

```
$ make
gcc -o hola_ejecutable hola.c -lm
$ ./hola_ejecutable
La raiz cuadrada de 16.000000 es 4.000000
$
```

Automáticamente se lanza el comando de compilación, y tras ello se puede ejecutar el binario generado.

Uso de variables en el Makefile

- Los Makefiles son scripts interpretables por el comando “make”.
- Permiten agrupar y parametrizar los comandos de compilación.
- El ejemplo presentado se puede hacer más genérico utilizando variables:

```
# Se define la variable EXEC_NAME con el nombre deseado para el ejecutable
EXEC_NAME=hola_ejecutable
```

```
# Se define la lista de ficheros fuente a compilar
SOURCE_FILES=hola.c
```

```
# Se definen los parámetros de enlazado
LDFLAGS=-lm
```

```
all:
    @echo Compilando y enlazando $(EXEC_NAME)...
    gcc -o $(EXEC_NAME) $(SOURCE_FILES) $(LDFLAGS)
```

El resultado al hacer “make”:

```
$ make
Compilando y enlazando hola_ejecutable...
gcc -o hola_ejecutable hola.c -lm
$
```



Makefile es una herramienta muy potente

Por ello se recomienda leer en detalle la siguiente documentación para aprender a manejarlos con soltura:

<http://www.chuidiang.com/clinux/herramientas/makefile.php>

Más conceptos a revisar

- Uso de Valgrind para debuggear problemas de memoria en el código (“punteros locos”, etc)
<http://arantxa.ii.uam.es/~talf1/valgrind.pdf>
- Overview del protocolo IRC:
<http://es.ccm.net/contents/703-irc>
https://es.wikipedia.org/wiki/Internet_Relay_Chat
- Más documentación en Moodle y
<http://metis.ii.uam.es/redes2/>