Herramientas de Desarrollo

Alejandro Furfaro

Abril 2016

Temario

Lenguajes

¿Que lenguaje hablan los microprocesadores?

Las CPU's definidas en los modelos originales fueron pensadas para tratar con valores que pueden tomar dos estados: Verdadero-Falso, 1 - 0, Tensión V - Tensión 0.
Por este motivo desde el inicio, cualquier Microprocesador solo "habla" en binario.
El problema es que a los seres humanos no nos resulta "natural" hablar ese lenguaje. Si bien podemos hacerlo, nos es engorroso, y por otra parte es muy fácil cometer un error. Basta con permutar un `1` con un `0` para tener un error. Y, una vez cometido,es sumamente arduo de encontrar.

Programando en el lenguaje del Microprocesador

El listado de la izquierda es el original. El de la derecha es una copia y tiene un error ¿donde está?

01101011	11011111	01101100	01101011	11011111	01101100
01000110	01110111	10001010	01000110	01110111	10001010
11101010	10010011	01101011	11101010	10010011	01101011
10100100	11010101	00110100	10100100	11010101	00110100
01100001	00010000	01101010	01100001	00010000	01101010
00011110	10001010	01011010	00011110	10001010	01011010
11010111	11010011	10100101	11010111	11010011	10100101
10001001	10010111	10011000	10001001	10010111	10011000
10001101	10100101	01111001	10001101	10100101	01111001
11000010	10010110	01101011	11000110	10010110	01101011
10110011	00101001	01111111	10110011	00101001	01111111
00101001	00010100	01101101	00101001	00010100	01101101
01010110	10010100	01100101	01010110	10010100	01100101

Programando en el lenguaje del Microprocesador

Y ? . . . ¿lo encontraste?
mmmm..... . . ¿estás seguro?

Programando en el lenguaje del Microprocesador

Y ? . . . ¿lo encontraste?
mmmm..... . . ¿estás seguro?

01101011	11011111	01101100	01101011	11011111	01101100
01000110	01110111	10001010	01000110	01110111	10001010
11101010	10010011	01101011	11101010	10010011	01101011
10100100	11010101	00110100	10100100	11010101	00110100
01100001	00010000	01101010	01100001	00010000	01101010
00011110	10001010	01011010	00011110	10001010	01011010
11010111	11010011	10100101	11010111	11010011	10100101
10001001	10010111	10011000	10001001	10010111	10011000
10001101	10100101	01111001	10001101	10100101	01111001
11000**0**10	10010110	01101011	11000**1**10	10010110	01101011
10110011	00101001	01111111	10110011	00101001	01111111
00101001	00010100	01101101	00101001	00010100	01101101
01010110	10010100	01100101	01010110	10010100	01100101

Necesitamos un lenguaje mas “humano”

```
GLOBAL main
EXTERN printf
; Constantes
LF      equ 0xA         ; 10 decimal
CR      equ 0xD         ; 13 decimal
NULL    equ 0           ; NULL
; Datos de lectura escritura
SECTION .data
zHola   db 'Hola Mundo', LF, CR, NULL
; Codigo
SECTION .text
main:
        push dword zHola; pusheamos direccion de zHola
        call printf     ; llamamos a printf
        add esp, 4      ; ajustamos la pila
        mov eax, 1      ; Nos preparamos....
        int 0x80        ; y nos vamos. Good bye
```

1º paso: Una sentencia = una instrucción

- Este es el lenguaje llamado Ensamblador, también conocido como “lenguaje de máquina”.
- Cada instrucción tiene un nombre alusivo a la operación que realiza (en inglés), y se lo representa por su abreviatura. Ej: MOV, por MOVE, ADD por ADDITION, etc.
- Cada sentencia en el programa corresponde a una y solo una instrucción de la CPU.
- Con ayuda de un programa llamado Ensamblador (o Assembler, igual que el lenguaje), se convierte ese texto, apto para su entendimiento por parte de los seres humanos, a números binarios, único lenguaje que habla el Microprocesador.
- Al texto original del programa escrito en lenguaje “humano” se lo conoce como ***código fuente***.

2º paso: Una sentencia = varias instrucciones

- A diferencia del Assembler, cada sentencia del programa se compone de varias instrucciones del procesador.
- La ventaja es que permite escribir aplicaciones de mayor complejidad son menos texto.
- El programa se escribe en un archivo de texto plano, igual que un programa en Assembler.
- Con ayuda de un programa llamado Compilador se convierte ese texto a números binarios, explotando cada sentencia en una o mas instrucciones del microprocesador.
- Al igual que el caso del programa escrito en Assembler, el texto escrito en C se denomina programa fuente. Obviamente esta denominación aplica al texto de cualquier lenguaje de programación.

El mismo programa anterior escrito en lenguaje C

```c
/* Esta secuencia es para iniciar un comentario.
El comentario puede ocupar cuantas lineas quieras
Y al final .....
Esta secuencia es para cerrar un comentario */

#include <stdio.h>

int     main ()
{
        printf("Hola Mundo!!\n");
        return 0;
}
```

¿Que contiene este simple programa?

1. En primer lugar lo mas fácil. Todo texto encerrado entre `/*` y `*/`, es tratado como un comentario. Significa que el compilador no va a generar código alguno con este texto.
2. Parece poco importante ya que no genera lógica ni agrega inteligencia al programa. Sin embargo los comentarios ayudan a explicar lo que estamos intentando hacer con nuestro algoritmo. Esto contribuye a la claridad de nuestro código, lo cual permite a otras personas o a nosotros mismos, modificar, corregir un defecto, o mejorar el programa con mayor facilidad. Incluir comentarios acertados y que agreguen claridad al código se considera una Buena Práctica de Programación.

¿Que contiene este simple programa?

3. Antes de continuar, aclaremos: Un programa C, se compone de dos elementos lógicos básicos: ***funciones*** y ***variables***. Las ***funciones*** contienen sentencias que definen las diferentes operaciones que se ejecutan una a una, y las ***variables*** contienen los datos que el programa mantiene almacenados, y modificará eventualemente como consecuencia de su operación.
4. Las funciones pueden llevar el nombre que mejor nos parezca, pero hay una función "obligatoria": `main`. Un programa comienza su ejecución en el inicio de la función `main`.

¿Que contiene este simple programa?

5. `main` para organizar el trabajo llama a otras funciones que como veremos van componiendo las partes que solucionan el problema completo (esto es programación modular).
6. Las funciones invocadas por `main` pueden estar escritas en el mismo archivo del programa, en otro archivo que junto con el nuestro componen el proyecto de software, o pueden ser funciones externas a nuestro programa que están guardadas en archivos que llamaremos bibliotecas de código, ya traducidas a números binarios, es decir en el lenguaje que entiende el microprocesador.

¿Que contiene este simple programa?

7. A continuación vemos la directiva

```
#include <stdio.h>
```

que le indica al compilador que debe incluir el archivo cabecera con las definiciones de las funciones de standard input output almacenadas en la biblioteca libc.

Concepto Importante

¡*stdio.h* no contiene el código de la biblioteca!. Es un archivo **de texto** en el que solamente se declaran las funciones que componen la biblioteca para que el compilador pueda conocer la sintaxis correcta para su invocación desde los programas. La biblioteca de código está en otro archivo (binario). El código fuente de las funciones que componen esta biblioteca, tampoco está en ***stdio.h***.
No olvidar este concepto.

¿Que contiene este simple programa?

8. Toda función puede recibir una lista de valores que se denominan ***argumentos***.
9. En el caso de `main` , en esta aplicación simple no recibe argumentos. Mas adelante en el curso veremos que puede recibirlos y como tratarlos en tal caso.
10. Luego entre los caracteres { y } se encierran las sentencias que componen el cuerpo de la función.
11. En el caso de este sencillo ejemplo el cuerpo de `main` solo contiene las sentencias:

```
printf("Hola Mundo!!\n");
return 0;
```

¿Que es printf?

12. No es otra cosa que una función.
13. Tal como explicamos recibe un argumento, en este caso el texto Hola Mundo!!\n
14. Lo que hace **`printf`** es imprimir en pantalla el texto que le pasamos como argumento.
15. \n es una secuencia de escape que utiliza el lenguaje C para representar el caracter Nueva Línea.
16. De este modo el comportamiento esperado de nuestro programa será imprimir en pantalla en el renglón siguiente al comando que lo ejecute, el mensaje Hola Mundo!!, y luego saltar a la línea siguiente como si se pulsase la tecla <Enter>
17. El tipo de argumento es una cadena de caracteres en forma de constante, por eso va encerrada entre comillas dobles.
18. A lo largo del curso vamos a utilizar mucho las cadenas de caracteres, de modo que es bueno empezar a familiarizarnos desde el principio.

¿Donde está printf?

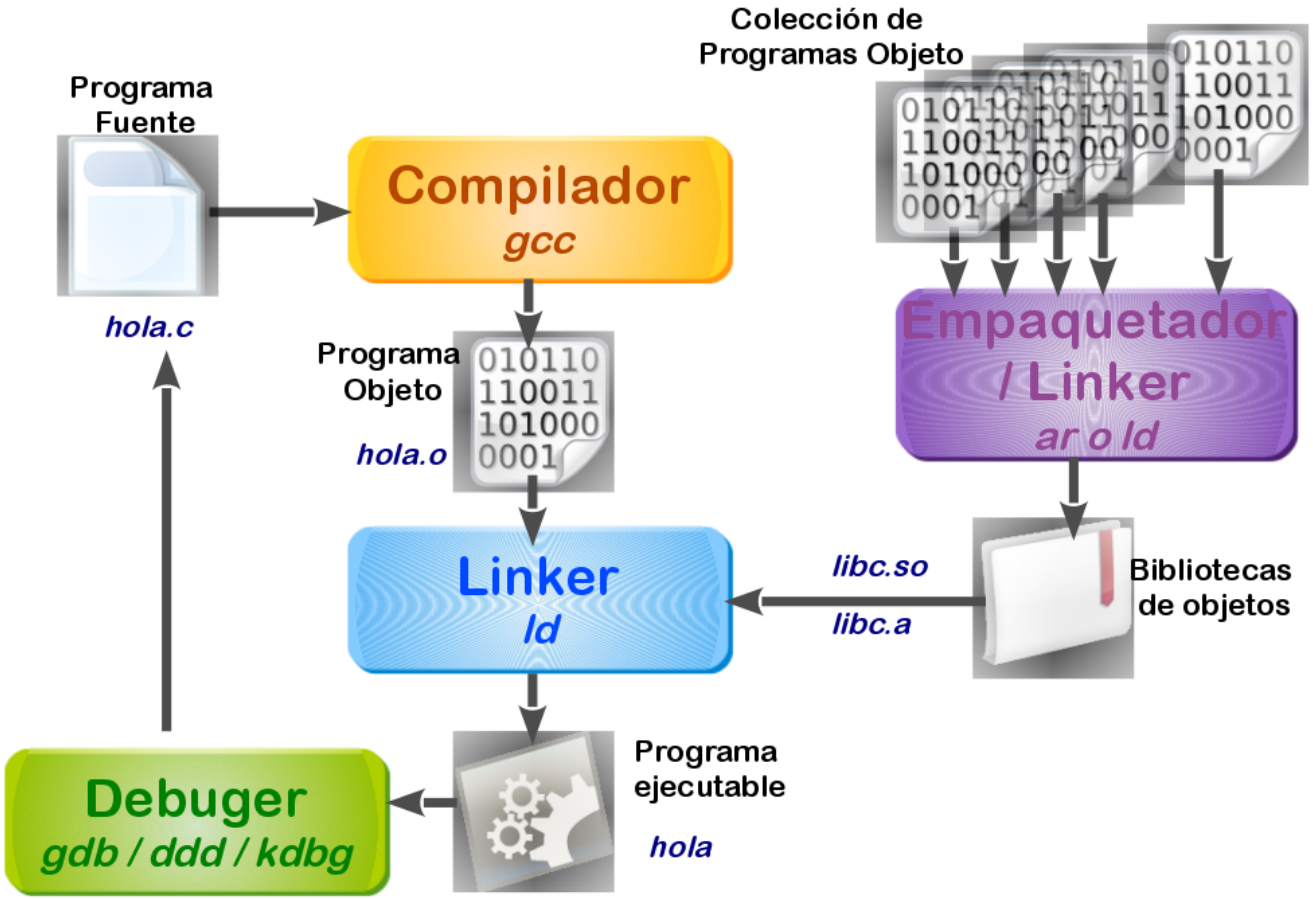
19. En nuestro archivo fuente, evidentemente no está.
20. De modo que solo cabe una posibilidad: La función es externa.
21. `printf` está contenida en una de las bibliotecas mas utilizadas en C: La de entrada salida estándar, cuyas definiciones estan en el archivo header stdio.h, ya explicado.
22. Comprobémoslo:

Tipear en la consola

```
locate stdio.h
grep ' printf ' /usr/include/stdio.h
```

23. Alguno de uds. estará preguntándose como se logra que el programa acceda al código de `printf` si ésta no es parte de programa sino que está afuera de él ¿verdad?
24. Quienes aun no se lo preguntaron... deberían hacerlo ;)

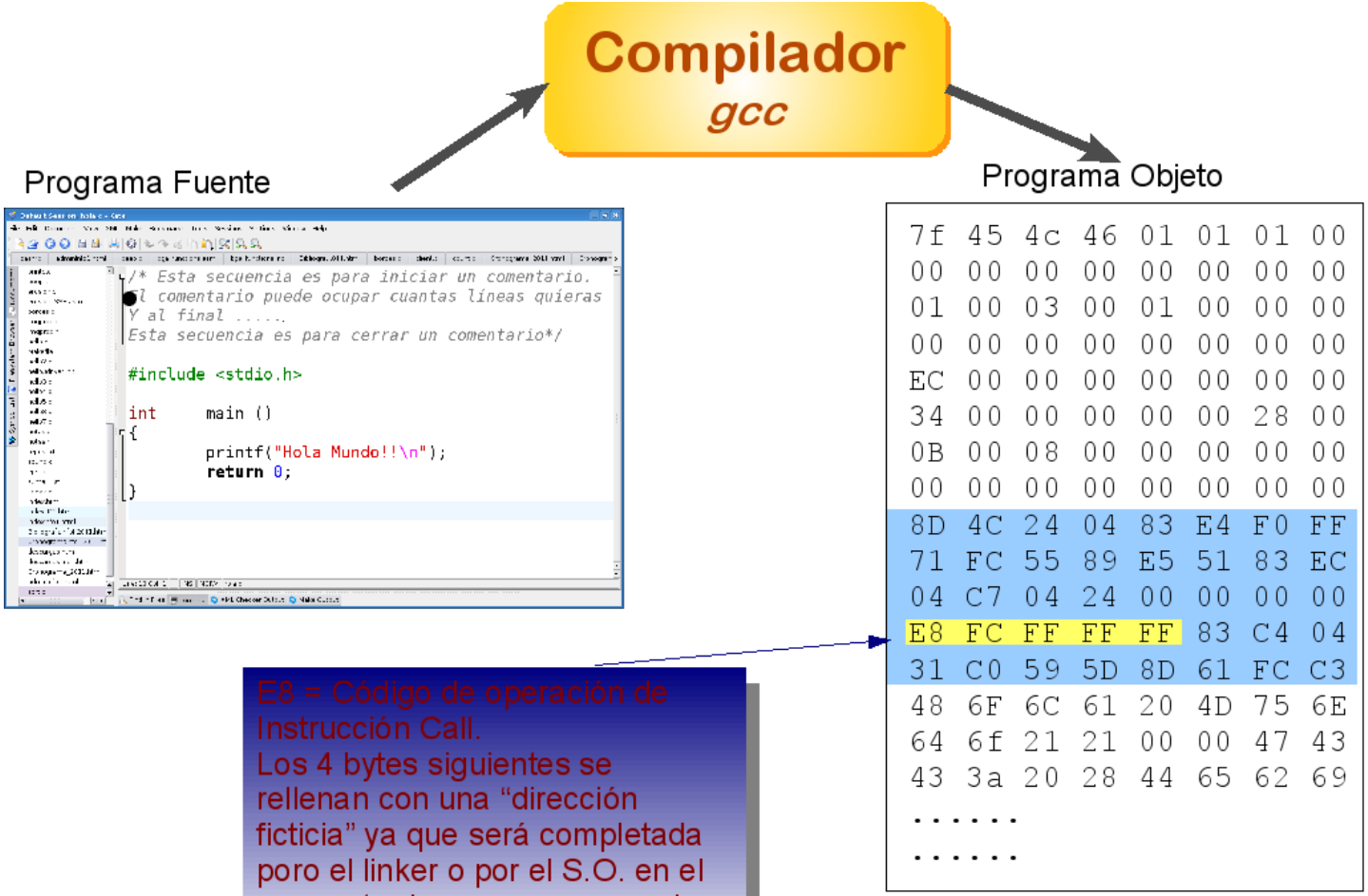
Proceso de desarrollo



El compilador

- Es un programa capaz de analizar sintácticamente un archivo de texto que contiene un programa fuente.
- Si éste está escrito de manera correcta, respetando la semántica del lenguaje para el cual compila, genera un código binario adecuado para ser ejecutado por el Microprocesador que obra como CPU en el sistema.
- Además de analizar las operaciones reemplaza los nombres lógicos que adoptemos en nuestro programa para variables o funciones por las direcciones de memoria en donde se ubican las mismas.
- No puede resolver referencias a funciones exteriores al archivo fuente que analiza. Por ejemplo, no puede resolver por que valor numérico reemplazar a la etiqueta `printf` , ya que no tiene visibilidad de la misma. Habrá que esperar a la siguiente fase para resolver este tema.

Cuando se dejan referencias por resolver



El compilador

- Antes de hacer su trabajo, invoca a un programa denominado preprocesador, que se encarga de eliminar los comentarios, incluir otros archivos (la línea #include <stdio.h>, es reemplazada por contenido del archivo stdio.h), y reemplaza las macros (la sentencia para el preprocesador en este caso es #define).
- Si genera errores el programa está mal escrito y debe ser revisado.
- Si no genera errores solo significa que el programa está correctamente escrito. De allí a que funcione correctamente es otra cuestión. . .
- Una vez que compiló, su producto es un programa objeto. Este es un binario pero que aún no está listo para poderse ejecutar.

Para generar el programa objeto, tipear en la consola

```
gcc -c hola.c -ohola.o -Wall
```

El compilador

- Antes de hacer su trabajo, invoca a un programa denominado preprocesador, que se encarga de eliminar los comentarios, incluir otros archivos (la línea #include <stdio.h>, es reemplazada por contenido del archivo stdio.h), y reemplaza las macros (la sentencia para el preprocesador en este caso es #define).
- Si genera errores el programa está mal escrito y debe ser revisado.
- Si no genera errores solo significa que el programa está correctamente escrito. De allí a que funcione correctamente es otra cuestión. . .
- Una vez que compiló, su producto es un programa objeto. Este es un binario pero que aún no está listo para poderse ejecutar.

Para generar el programa objeto, tipear en la consola

```
gcc -c hola.c -ohola.o -Wall
```

El Linker

- Es un programa capaz de tomar el programa objeto generado recién por el compilador, enlazarlo (“linkearlo”) con otros programas objeto y con otras biblioteca de código y generar un programa ejecutable por el Sistema Operativo sobre el cual estamos desarrollando nuestro programa.
- Muchas cosas juntas ¿verdad?
- Enlazar significa:
  - Poner todos los bloques de código juntos y ordenar código y datos en secciones comunes para luego guardar ese conjunto en un único archivo ejecutable.
  - Una vez ordenado, resolver cada referencia a una variable o función que en la fase de compilación eran externas. En nuestro caso el linker resolverá la referencia a `printf` .
  - Identificar y marcar el punto de entrada del programa (la dirección que se le asignará a `main`).

El linker

- Parece poco relevante. Sin embargo es crucial esta fase de la generación de nuestro programa

Para generar el programa ejecutable podríamos, tipear en la consola

```
ld --eh-frame-hdr -m elf_i386 --hash-style=both
-dynamic-linker /lib/ld-linux.so.2 -o hola /usr/lib/crt1.o
/usr/lib/crti.o /usr/lib/gcc/i486-linux-gnu/4.3.2/crtbegin.o
-L/usr/lib/gcc/i486-linux-gnu/4.3.2 -L/usr/lib hola.o -lgcc
--as-needed -lgcc_s --no-as-needed -lc -lgcc --as-needed
-lgcc_s --no-as-needed /usr/lib/gcc/i486-linux-gnu/4.3.2/crtend.o
/usr/lib/crtn.o
```

- Hay involucrados unos cuantos objetos como vemos que son relevantes: crt1.o, crti.o, crtbegin.o, crtend.o.
- Y algún que otro componente adicional.
- Engorroso, imposible de memorizar, y sobre todo, sujeto a cuestiones internas del sistema.

El linker

- Parece poco relevante. Sin embargo es crucial esta fase de la generación de nuestro programa

Para generar el programa ejecutable podríamos, tipear en la consola

```
ld --eh-frame-hdr -m elf_i386 --hash-style=both
-dynamic-linker /lib/ld-linux.so.2 -o hola /usr/lib/crt1.o
/usr/lib/crti.o /usr/lib/gcc/i486-linux-gnu/4.3.2/crtbegin.o
-L/usr/lib/gcc/i486-linux-gnu/4.3.2 -L/usr/lib hola.o -lgcc
--as-needed -lgcc_s --no-as-needed -lc -lgcc --as-needed
-lgcc_s --no-as-needed /usr/lib/gcc/i486-linux-gnu/4.3.2/crtend.o
/usr/lib/crtn.o
```

- Hay involucrados unos cuantos objetos como vemos que son relevantes: crt1.o, crti.o, crtbegin.o, crtend.o.
- Y algún que otro componente adicional.
- Engorroso, imposible de memorizar, y sobre todo, sujeto a cuestiones internas del sistema.

El linker

- Parece poco relevante. Sin embargo es crucial esta fase de la generación de nuestro programa

Para generar el programa ejecutable podríamos, tipear en la consola

```
ld --eh-frame-hdr -m elf_i386 --hash-style=both
-dynamic-linker /lib/ld-linux.so.2 -o hola /usr/lib/crt1.o
/usr/lib/crti.o /usr/lib/gcc/i486-linux-gnu/4.3.2/crtbegin.o
-L/usr/lib/gcc/i486-linux-gnu/4.3.2 -L/usr/lib hola.o -lgcc
--as-needed -lgcc_s --no-as-needed -lc -lgcc --as-needed
-lgcc_s --no-as-needed /usr/lib/gcc/i486-linux-gnu/4.3.2/crtend.o
/usr/lib/crtn.o
```

- Hay involucrados unos cuantos objetos como vemos que son relevantes: crt1.o, crti.o, crtbegin.o, crtend.o.
- Y algún que otro componente adicional.
- Engorroso, imposible de memorizar, y sobre todo, sujeto a cuestiones internas del sistema.

El linker

- Por eso, gcc sabe llamar al linker y nos evita este engorroso trámite a nosotros

Para generar el programa ejecutable tipeamos en la consola

```
gcc -ohola hola.o -Wall
```

- Para saber como el gcc arma el llamado usamos la opción -v (verbose)

Tipear en la consola

```
gcc -ohola hola.o -v
```

El linker

- Por eso, gcc sabe llamar al linker y nos evita este engorroso trámite a nosotros

Para generar el programa ejecutable tipeamos en la consola

```
gcc -ohola hola.o -Wall
```

- Para saber como el gcc arma el llamado usamos la opción -v (verbose)

Tipear en la consola

```
gcc -ohola hola.o -v
```

El linker

- Por eso, gcc sabe llamar al linker y nos evita este engorroso trámite a nosotros

Para generar el programa ejecutable tipeamos en la consola

```
gcc -ohola hola.o -Wall
```

- Para saber como el gcc arma el llamado usamos la opción -v (verbose)

Tipear en la consola

```
gcc -ohola hola.o -v
```

El linker

- Por eso, gcc sabe llamar al linker y nos evita este engorroso trámite a nosotros

Para generar el programa ejecutable tipeamos en la consola

```
gcc -ohola hola.o -Wall
```

- Para saber como el gcc arma el llamado usamos la opción -v (verbose)

Tipear en la consola

```
gcc -ohola hola.o -v
```

El linker

- Por eso, gcc sabe llamar al linker y nos evita este engorroso trámite a nosotros

Para generar el programa ejecutable tipeamos en la consola

```
gcc -ohola hola.o -Wall
```

- Para saber como el gcc arma el llamado usamos la opción -v (verbose)

Tipear en la consola

```
gcc -ohola hola.o -v
```

Warning: Prestale atención a los warnings

- Los Warnings que arrojan tanto el Compilador como el linker, son commo su nombre lo indica Ardvertencias.
- No impiden que el compilador genere el archivo con el objeto reubicable ni que el linker genere el archivo ejecutable.
- No por eso debemos ignorarlos.
- Por el contrario debe prestarse especial atención a los Warnings
- La experiencia indica que terminan transformándose en errores de lógica.
- La opción `-Wall` en ambas líneas de compilación y linkeo, indica a ambas herramientas que presenten Todos los Warnings (`Warning all`). Aun los mas insignificantes

Warning: Prestale atención a los warnings

- Los Warnings que arrojan tanto el Compilador como el linker, son commo su nombre lo indica Ardvertencias.
- No impiden que el compilador genere el archivo con el objeto reubicable ni que el linker genere el archivo ejecutable.
- No por eso debemos ignorarlos.
- Por el contrario debe prestarse especial atención a los Warnings
- La experiencia indica que terminan transformándose en errores de lógica.
- La opción `-Wall` en ambas líneas de compilación y linkeo, indica a ambas herramientas que presenten Todos los Warnings (`Warning all`). Aun los mas insignificantes

Warning: Prestale atención a los warnings

- Los Warnings que arrojan tanto el Compilador como el linker, son commo su nombre lo indica Ardvertencias.
- No impiden que el compilador genere el archivo con el objeto reubicable ni que el linker genere el archivo ejecutable.
- No por eso debemos ignorarlos.
- Por el contrario debe prestarse especial atención a los Warnings
- La experiencia indica que terminan transformándose en errores de lógica.
- La opción `-Wall` en ambas líneas de compilación y linkeo, indica a ambas herramientas que presenten Todos los Warnings (`Warning all`). Aun los mas insignificantes

Warning: Prestale atención a los warnings

- Los Warnings que arrojan tanto el Compilador como el linker, son commo su nombre lo indica Ardvertencias.
- No impiden que el compilador genere el archivo con el objeto reubicable ni que el linker genere el archivo ejecutable.
- No por eso debemos ignorarlos.
- Por el contrario debe prestarse especial atención a los Warnings
- La experiencia indica que terminan transformándose en errores de lógica.
- La opción `-Wall` en ambas líneas de compilación y linkeo, indica a ambas herramientas que presenten Todos los Warnings (`Warning all`). Aun los mas insignificantes

Warning: Prestale atención a los warnings

- Los Warnings que arrojan tanto el Compilador como el linker, son commo su nombre lo indica Ardvertencias.
- No impiden que el compilador genere el archivo con el objeto reubicable ni que el linker genere el archivo ejecutable.
- No por eso debemos ignorarlos.
- Por el contrario debe prestarse especial atención a los Warnings
- La experiencia indica que terminan transformándose en errores de lógica.
- La opción `-Wall` en ambas líneas de compilación y linkeo, indica a ambas herramientas que presenten Todos los Warnings (`Warning all`). Aun los mas insignificantes

Warning: Prestale atención a los warnings

- Los Warnings que arrojan tanto el Compilador como el linker, son commo su nombre lo indica Ardvertencias.
- No impiden que el compilador genere el archivo con el objeto reubicable ni que el linker genere el archivo ejecutable.
- No por eso debemos ignorarlos.
- Por el contrario debe prestarse especial atención a los Warnings
- La experiencia indica que terminan transformándose en errores de lógica.
- La opción `-Wall` en ambas líneas de compilación y linkeo, indica a ambas herramientas que presenten Todos los Warnings (`Warning all`). Aun los mas insignificantes

Buenas Costumbres

Buena Práctica de Desarrollo

Siempre incluir la opción `-Wall` en las líneas de compilación y linkeo, trabajando sobre el código para eliminar **TODOS** los warnings.
Algo es 100 % seguro: ***El Warning se transforma en un bug mas tarde o mas temprano***.
Por lo tanto se deberá incluir en cada proyecto `-Wall` en las líneas de compilación y linkeo **SIEMPRE**.

Agreguemos alguna función de cálculo

```
/* Programa sqrt.c:
*  Su funci n es calcular la ra z cuadrada de un n mero
*  predefinido en su c digo y mostrar su resultado en
*  la pantalla del computador.
*  Para compilarlo: gcc -c sqrt.c -o sqrt.o
*  Para linkearlo: gcc sqrt.o -o sqrt -lm
**/

#include <stdio.h>
#include <math.h>

#define N 1234567890

int main ()
{
        double result;
        result = sqrt(N);
        printf ("La raiz cuadrada de %d es:%10.7f\n",N,result);
        return 0;
}
```

Linkeando con una Biblioteca

- Si observamos el comentario que encabeza el listado del programa del slide anterior, vemos que al linker se le provee una opción adicional: `-lm`
- `-l` sirve para especificar el nombre de una Biblioteca (l por library)
- m es el nombre de la biblioteca: `m` es math, cuyos prototipos, macros y constantes están definidos en math.h (entre ellos la función `sqrt`)
- Pregunta: ¿Porque no hubo que especificar la librería que contiene `printf` ?
- El compilador “conoce” la ubicación de las bibliotecas mas comunes para evitar que debamos especificar permanentemente librerías de uso casi tan común como la propia función `main`

Linkeando con una Biblioteca

- Si observamos el comentario que encabeza el listado del programa del slide anterior, vemos que al linker se le provee una opción adicional: `-lm`
- `-l` sirve para especificar el nombre de una Biblioteca (l por library)
- m es el nombre de la biblioteca: `m` es math, cuyos prototipos, macros y constantes están definidos en math.h (entre ellos la función `sqrt`)
- Pregunta: ¿Porque no hubo que especificar la librería que contiene `printf` ?
- El compilador “conoce” la ubicación de las bibliotecas mas comunes para evitar que debamos especificar permanentemente librerías de uso casi tan común como la propia función `main`

Linkeando con una Biblioteca

- Si observamos el comentario que encabeza el listado del programa del slide anterior, vemos que al linker se le provee una opción adicional: `-lm`
- `-l` sirve para especificar el nombre de una Biblioteca (l por library)
- m es el nombre de la biblioteca: `m` es math, cuyos prototipos, macros y constantes están definidos en math.h (entre ellos la función `sqrt`)
- Pregunta: ¿Porque no hubo que especificar la librería que contiene `printf` ?
- El compilador “conoce” la ubicación de las bibliotecas mas comunes para evitar que debamos especificar permanentemente librerías de uso casi tan común como la propia función `main`

Que Aprendimos?

- Que son y que relación tienen los diferentes lenguajes, binario, assembler, C.
- Las herramientas de desarrollo que utilizamos para construir programas, su uso y conceptos.
- Hicimos algunos ejemplos para empezar a caminar.
- Ahora vamos a mejorarlos y aumentar sus posibilidades