Una Introducción a GCC

para los Compiladores de GNU `gcc` y `g++`
Revisado y actualizado

Brian Gough
Prefacio por Richard M. Stallman

El registro de este libro está disponible en la Biblioteca Británica.

Segunda impresión, Agosto 2005 (1/8/2005). Revisado y actualizado.
Primera impresión, Marzo 2004 (7/3/2004).
Traducido por David Arroyo Menéndez y Luis Palomo de Onís Gutiérrez, Agosto 2011 (10/8/2011). Revisado y actualizado (gcc 4.6).
Publicado por Network Theory Limited.15 Royal Park
Bristol
BS8 3AL
United Kingdom

Correo electrónico: info@network-theory.co.uk

ISBN 0-9541617-9-3

Hay más información de este libro disponible desde
http://www.network-theory.co.uk/gcc/intro/

Imagen de cubierta: Del diseño de una pila de hardware rápida y eficiente energéticamente.[1] Imagen creada con el sistema de diseño libre Electric VLSI escrito por Steven Rubin de Static Free Software (www.staticfreesoft.com). Static Free Software proporciona soporte para Electric en la industria del diseño electrónico

[1] "A Fast and Energy-Efficient Stack" por J. Ebergen, D. Finchelstein, R. Kao, J. Lexau y R. Hopkins.

Índice General

Una Introducción a GCC **1**

Prefacio **3**

1 Introducción **5**
1.1 Una breve historia de GCC 5
1.2 Importantes características de GCC 6
1.3 Programación en C y C++ 7
1.4 Convenciones usadas en este manual 7

2 Compilando un programa C **9**
2.1 Compilando un pequeño programa C 9
2.2 Encontrando errores en un pequeño programa 10
2.3 Compilando múltiples archivos fuentes 11
2.4 Compilando archivos independientes 13
2.4.1 Creando archivos objeto desde archivos fuente ... 13
2.4.2 Creando ejecutables desde archivos objeto 14
2.5 Recompilando y reenlazando 15
2.6 Un pequeño makefile 16
2.7 Enlazando con librerías externas 18
2.7.1 Orden de enlace de librerías 20
2.8 Usando librerías de archivos de cabeceras 21

3 Opciones de compilación **23**
3.1 Asignando rutas de búsqueda 23
3.1.1 Ejemplo de ruta de búsqueda 24
3.1.2 Variables de entorno 26
3.1.3 Rutas de búsqueda extendidas 27
3.2 Librerías compartidas y librerías estáticas 28
3.3 Estándares del lenguaje C 31
3.3.1 ANSI/ISO 31
3.3.2 ANSI/ISO estricto 33
3.3.3 Seleccionando estándares específicos 34
3.4 Opciones de aviso en `-Wall` 35
3.5 Opciones de aviso adicionales 37
3.6 Opciones de aviso recomendadas 41

4 Usando el preprocesador 43
4.1 Definiendo macros 43
4.2 Macros con valor 44
4.3 Preprocesando archivos fuentes 46

5 Compilando para depuración 49
5.1 Examinando archivos core 49
5.2 Mostrando un rastreo 52
5.3 Poniendo un punto de ruptura 53
5.4 Paso a paso a través de un programa 53
5.5 Modificando variables 54
5.6 Continuando la ejecución 54
5.7 Más información 55

6 Compilando con optimización 57
6.1 Optimización a nivel de fuentes 57
6.1.1 Eliminación de subexpresión común 57
6.1.2 Expansión de función en línea 58
6.2 Dilema velocidad-espacio 60
6.2.1 Desenrollado de bucle 60
6.3 Planificación 61
6.4 Niveles de optimización 62
6.5 Ejemplos 63
6.6 Optimización y depuración 66
6.7 Optimización y avisos del compilador 66

7 Compilando un programa C++ 69
7.1 Compilando un pequeño programa C++ 69
7.2 Opciones de compilación en C++ 71
7.3 Usando la librería estándar de C++ 72
7.4 Plantillas 72
7.4.1 Usando plantillas de librerías estándar de C++ 73
7.4.2 Proporcionando sus propias plantillas 74
7.4.3 Instanciación explícita de plantillas 76
7.4.4 La palabra reservada `export` 77

8 Opciones específicas de plataforma 79

8.1 Opciones para Intel y AMD x86 79
8.1.1 Extensiones x86 80
8.1.2 Procesadores x86 de 64 bits 81
8.2 Opciones para DEC Alpha 81
8.3 Opciones para SPARC 83
8.4 Opciones para POWER/PowerPC 83
8.5 Soporte de múltiples arquitecturas 84
8.6 Usos de coma flotante 84
8.7 Portabilidad de los tipos con signo y sin signo 87

9 Resolución de problemas 91

9.1 Opciones de ayuda en línea de comandos 91
9.2 Números de versión 91
9.3 Compilación verbosa 92
9.4 Parando un programa en un bucle infinito 95
9.5 Previniendo un uso excesivo de memoria 97

10 Utilidades relativas al compilador 99

10.1 Creando una librería con el archivador de GNU 99
10.2 Usando el profiler `gprof` 101
10.3 Test de cobertura con `gcov` 104

11 Como funciona el compilador 107

11.1 Una vista de los procesos de compilación 107
11.2 El preprocesador 108
11.3 El compilador 108
11.4 El ensamblador 109
11.5 El enlazador 109

12 Examinado archivos compilados 111

12.1 Identificando archivos 111
12.2 Examinando la tabla de símbolos 112
12.3 Encontrando librerías dinámicas enlazadas 113

13 Mensajes comunes de error 115

13.1 Mensajes de error del preprocesador 115
13.2 Mensajes de error del compilador 116
13.3 Mensajes de error del enlazador 125
13.4 Mensajes de error en tiempo de ejecución 127

14 Obteniendo ayuda **129**

Lectura adicional **131**

Reconocimientos **133**

Organizaciones de software libre **135**

Licencia para documentación libre GNU **137**

ADDENDUM: How to use this License for your documents 142

Índice **143**

Una Introducción a GCC

Este manual proporciona una introducción a los Compiladores de GNU de C y C++, `gcc` y `g++`, que son parte de la Colección de Compiladores de GNU (GCC).

El desarrollo de este manual fué realizado enteramente por Network Theory Ltd. Las copias publicadas por Network Theory Ltd traerán dinero para más documentación libre.

Prefacio

Este Prefacio es una amable contribución de Richard M. Stallman, el principal autor de GCC y fundador del Proyecto GNU.

Este libro es una guía para iniciarse en GCC, GNU Compiler Collection (Colección de Compiladores GNU). Se mostrará cómo usar GCC como una herramienta de programación. GCC es una herramienta de programación, esto es verdad— pero también es algo más. También forma parte de la campaña por la libertad de los usuarios de ordenadores desde hace más de 20 años.

Todo lo que queremos es buen software, pero ¿qué significa para nosotros que un software sea bueno?. Funcionalidades adecuadas y fiabilidad puede ser algo *técnicamente* bueno, pero esto no es suficiente. Un buen software debe además ser *éticamente* bueno: tiene que ser respetuoso con la libertad de los usuarios.

Como usuario de software, se debería tener el derecho a ejecutarlo como se necesite, el derecho a estudiar el código fuente y a cambiarlo como se desee, el derecho a redistribuir copias de éste a terceras personas, y el derecho a publicar versiones modificadas con lo que se puede contribuir a la construcción de la comunidad. Cuando un programa respeta la libertad de esta forma, se dice que es *software libre*. Anteriormente a GCC había otros compiladores para C, Fortran, Ada, etc. Pero no eran software libre, y no se podían usar libremente. Escribí el GCC para que se pueda usar un compilador sin perder nuestra libertad.

Un compilador solo no es suficiente —para usar un sistema de computación, se debe disponer de un sistema operativo completo. En 1983, todos los sistemas operativos para ordenadores modernos eran no libres. Para remediar esto, en 1984 comencé a desarrollar el sistema operativo GNU, un sistema similiar a Unix que sería software libre. El desarrollo de GCC fué una parte del desarrollo de GNU.

A principios de los 90, el recién terminado sistema operativo GNU fué completado con la suma de un kernel, Linux, que se hizo software libre en 1992. El sistema operativo combinado GNU/Linux ha alcanzado la meta de hacer posible el uso de una computadora en libertad. Pero la libertad nunca está automáticamente asegurada, y debemos trabajar para protegerla. El Movimiento del Software Libre necesita tu apoyo.

Richard M. Stallman
Febrero de 2004

1 Introducción

El propósito de este libro es explicar el uso de los compiladores de GNU C y C++, `gcc` y `g++`. Después de leer este libro se comprenderá como compilar un programa y, cómo usar las opciones básicas del compilador para optimización y depuración. Este libro no intenta enseñar los lenguajes C o C++ en sí, este material puede ser encontrado en muchos otros lugares (véase [Lectura adicional], página 131).

Los programadores experimentados que están familiarizados con otros sistemas, pero son nuevos en compiladores GNU, pueden saltarse las primeras secciones de los capítulos "*Compilando un programa C*", "*Usando el preprocesador*" y "*Compilando un programa C++*". Las secciones y capítulos restantes proporcionan una buena visión del conjunto de las funcionalidades de GCC para aquellos que ya saben cómo usar otros compiladores.

1.1 Una breve historia de GCC

El autor original del Compilador de C de GNU (GCC) es Richard Stallman, el fundador del Proyecto GNU.

El Proyecto GNU fué iniciado en 1984 para crear un sistema operativo basado en software libre similar a UNIX y, así promover la libertad y la cooperación entre usarios de ordenadores y programadores. Cualquier sistema operativo basado en UNIX necesita un compilador de C, y no había compiladores libres en ese momento, el Proyecto GNU debía desarrollar uno desde cero. Este trabajo fué financiado por donaciones de individuos y compañias a través de la Free Software Foundation, una organización sin ánimo de lucro destinada a dar soporte al trabajo del Proyecto GNU.

La primera entrega de GCC fué hecha en 1987. Esto fué un significativo progreso, siendo el primer compilador portable para optimizar ANSI C liberado como software libre. Desde este momento GCC ha llegado a ser uno de las más importantes herramientas en el desarrollo de software libre.

Un avance importante en el compilador llega con la serie 2.0 en 1992, que añade la capacidad de compilar C++. En 1997, se creó una rama experimental del compilador (EGCS) para mejorar la optimización y el soporte de C++. Después de este trabajo, EGCS fué adoptado como la principal línea de desarrollo de GCC y, estas funcionalidades llegaron a estar ampliamente disponibles en la versión 3.0 de GCC en 2001.

A través del tiempo GCC ha sido extendido para dar soporte a muchos lenguajes adicionales, incluyendo Fortran, ADA, Java y Objective-C. El acrónimo GCC es ahora usado para referir al "GNU Compiler Collection" (Colección de Compiladores de GNU). Su desarrollo está guiado por el *GCC Steering Committee*, un grupo compuesto de representantes de las comunidades de usuarios/as de GCC en la industria, la investigación y la academia.

1.2 Importantes características de GCC

Esta sección describe algunas de las más importantes funcionalidades de GCC.

Lo primero de todo, GCC es un compilador portable —se ejecuta en la mayoría de las plataformas disponibles hoy, y puede producir salidas para muchos tipos de procesadores. Además de procesadores usados en ordenadores personales, también soporta microcontroladores, DSPs y CPUs de 64 bits.

GCC no es solo un compilador nativo —también puede *compilar cruzado* cualquier programa, produciendo ficheros ejecutables para un sistema diferente desde el que GCC está siendo usado. Esto permite compilar software para sistemas embebidos que no son capaces de ejecutar un compilador. GCC está escrito en C con un fuerte enfoque hacia la portabilidad, y puede compilarse a sí mismo, así puede ser adaptado a nuevos sistemas fácilmente.

GCC tiene múltiples *frontends*, para parsear diferentes lenguajes. Los programas en cada lenguaje pueden ser compilados, o compilados de manera cruzada, para cualquier arquitectura. Por ejemplo, un programa en ADA puede ser compilado para un microcontrolador, o un programa en C para un supercomputador.

GCC tiene un diseño modular, permitiendo que el soporte para nuevos lenguajes y arquitecturas sea añadido. Añadir un nuevo front-end a GCC habilita el uso de este lenguaje en cualquier arquitectura y proporciona que estén disponibles facilidades (tales como librerías) en tiempo de ejecución. De manera similar, si se añade soporte para una nueva arquitectura éste se vuelve disponible para todos los lenguajes.

Finalmente, y de manera más importante, GCC es software libre, distribuido bajo la GNU General Public License (GNU GPL).[1] Esto significa que se tiene la libertad para usar y modificar GCC, como con todo el software de GNU. Si se necesita soporte para un nuevo tipo de CPU, un nuevo lenguaje, o una nueva funcionalidad

[1] Para detalles ver el fichero de licencia '`COPYING`' distribuido con GCC.

es posible añadirla uno mismo o contratar a alguien para mejorar GCC de manera personalizada. Se puede contratar a alguien para arreglar un error si esto es importante en el trabajo cotidiano.

Más allá, hay libertad para compartir cualquier mejora hecha a GCC. Como resultado de esta libertad, se pueden usar las mejoras hechas a GCC por otras personas. Las muchas funcionalidades ofrecidas por GCC hoy muestran cómo esta libertad de cooperar funciona en tu beneficio, y en el de cualquiera que use GCC.

1.3 Programación en C y C++

C y C++ son lenguajes que permiten acceso directo a la memoria del ordenador. Históricamente, han sido usados para escribir sistemas software de bajo nivel, y aplicaciones dónde el alto rendimiento o el control a través del uso de recursos son críticos. Sin embargo, se requiere de gran cuidado para asegurar que la memoria es accedida de manera correcta, para evitar la corrupción de otras estructuras de datos. Este libro describe técnicas que ayudarán a detectar potenciales errores durante la compilación, pero los riesgos de usar lenguajes como C o C++ nunca pueden ser eliminados.

Además de C y C++ el Proyecto GNU también proporciona otros lenguajes de alto nivel, tales como GNU Common Lisp (`gcl`), GNU Smalltalk (`gst`), el GNU Scheme extension language (`guile`) y el GNU Compiler para Java (`gcj`). Estos lenguajes no permiten al usuario acceder a memoria directamente, eliminando la posibilidad de errores de acceso a memoria. Son una alternativa segura a C y C++ para muchas aplicaciones.

1.4 Convenciones usadas en este manual

Este manual contiene muchos ejemplos que pueden ser escritos en el teclado. Un comando introducido en el terminal se muestra como esto,

```
$ comando
```

seguido por su salida. Por ejemplo:

```
$ echo "hola mundo"
hola mundo
```

El primer carácter en la línea es el prompt del terminal, y no será escrito. El signo del dólar '`$`' es usado como el prompt estándar en este manual, aunque en algunos sistemas puede usar un carácter diferente.

Cuando un comando en un ejemplo es demasiado largo para ajustarse en una sola línea es envuelto e indentado en las líneas subsiguientes, como este:

```
$ echo "un ejemplo de una línea que es demasiado
    larga para este manual"
```

Cuando se introduce en el teclado, el comando entero será escrito en una sola línea.

Los ficheros fuente de ejemplo usados en este manual pueden ser descargados desde el sitio web de la editorial,[2] o introducidos a mano usando cualquier editor de texto, tal como el editor estándar de GNU, `emacs`. Los comandos de compilación de ejemplo usan `gcc` y `g++` como los nombres de los compiladores de GNU de C y de C++ y usan `cc` para referirse a otros compiladores. Los programas de ejemplo trabajarán con cualquier versión de GCC. Las opciones de estos comandos que están disponibles en versiones recientes de GCC son anotadas en el texto.

Este ejemplo asume el uso de un sistema operativo GNU —hay pequeñas diferencias en la salida en otros sistemas. Algunos mensajes de salida no esenciales y dependientes del sistema (tal como rutas muy largas) han sido editadas en los ejemplos por brevedad. Los comandos para configurar variables de entorno usan la sintaxis de la shell estándar de GNU (`bash`), y funcionará con cualquier versión de Bourne shell.

[2] Ver http://www.network-theory.co.uk/gcc/intro/

2 Compilando un programa C

Este capítulo describe cómo compilar programas C usando `gcc`. Los programas pueden ser compilados desde un solo fichero fuente o desde múltiples ficheros fuente, y pueden usar librerías de sistema y ficheros de cabecera.

La compilación se refiere al proceso de convertir un programa desde el *código fuente* textual de un lenguaje de programación tal como C o C++, en *código máquina*, la secuencia de unos y ceros usados para controlar la unidad central de proceso (CPU) del ordenador. Este código máquina es almacenado en un fichero conocido como *fichero ejecutable*, a veces también llamado *fichero binario*.

2.1 Compilando un pequeño programa C

El clásico programa de ejemplo para el lenguaje C es *Hola Mundo*. Aquí está el código fuente para nuestra versión del programa:

```
#include <stdio.h>

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

Se asume que el código fuente está almacenado en un fichero llamado '`hola.es.c`'. Para compilar el fichero '`hola.es.c`' con `gcc`, se puede usar el siguiente comando:

```
$ gcc -Wall hola.es.c -o hola
```

Esto compila el código fuente de '`hola.es.c`' a código máquina y lo almacena en el fichero ejecutable '`hola`'. El fichero de salida para el código máquina se especifica usando la opción '`-o`'. Esta opción normalmente es el último argumento en la línea de comandos. Si se omite, la salida es escrita a un fichero por defecto llamado '`a.out`'.

Nótese que si ya existe un fichero con el mismo nombre que el fichero ejecutable en el directorio actual, entonces se sobreescribirá.

La opción '`-Wall`' activa todos los avisos más comunes —**¡se recomienda usar siempre esta opción!**. Hay muchas otras opciones de avisos que serán discutidas en capítulos posteriores, pero '`-Wall`' es la más importante. GCC no producirá avisos a menos que estén activados. Los avisos del compilador son una ayuda esencial detectando problemas al programar en C y C++.

En este caso, el compilador no produce avisos con la opción '`-Wall`', debido a que el programa es completamente válido. El código fuente que no produce avisos se dice que *compila limpiamente*.

Para ejecutar el programa, escribe la ruta del ejecutable así:

```
$ ./hola
¡Hola, mundo!
```

Esto carga el fichero ejecutable en memoria y hace que la CPU empiece a ejecutar las instrucciones que contiene. La ruta `./` se refiere al directorio actual, así `./hola` carga y ejecuta el fichero ejecutable '`hola`' localizado en el directorio actual.

2.2 Encontrando errores en un pequeño programa

Como se mencionó antes, los avisos del compilador son una ayuda esencial cuando se programa en C y C++. Para demostrar esto, el programa de debajo contiene un pequeño error: usa la función `printf` de manera incorrecta, especificando un formato en coma flotante '`%f`' para un valor entero:

```
#include <stdio.h>

int
main (void)
{
  printf ("Dos y dos son %f\n", 4);
  return 0;
}
```

Este error no es obvio a primera vista, pero puede ser detectado por el compilador si la opción de aviso '`-Wall`' se ha habilitado.

Al compilar el anterior programa, '`mal.es.c`', con la opción '`-Wall`' produce el siguiente mensaje:

```
$ gcc -Wall bad.c -o bad
bad.c: In function 'main':
bad.c:6:3: warning: format '%f' expects argument of
type 'double', but argument 2 has type 'int' [-Wformat]
```

Esto indica que un formato de cadena ha sido usado incorrectamente en el fichero '`mal.es.c`' en la línea 6. Los mensajes producidos por GCC siempre tienen la forma *fichero:número de línea:mensaje*. El compilador distingue entre *mensajes de error*, que impiden una compilación exitosa, y *mensajes de aviso* que indican posibles problemas (pero no detienen la compilación).

En este caso, el especificador de formato será '`%d`' para un argumento entero. Los especificadores de formato permitidos para `printf` pueden ser encontrados en cualquier libro general de C, tal como el *GNU C Library Reference Manual* (véase [Lectura adicional], página 131).

Sin la opción de aviso '`-Wall`' el programa compila limpiamente, pero produce resultados incorrectos:

```
$ gcc mal.es.c -o mal
$ ./mal
Dos y dos son 0.000000      (salida incorrecta)
```

El incorrecto formato especificado causa que la salida esté corrompida, porque a la función `printf` se le pasa un entero en lugar de un número en coma flotante. Los números enteros y en coma flotante son almacenados en diferentes formatos en memoria, por lo general ocupan diferente número de bytes, obteniendo un falso resultado. La actual salida que se muestra puede diferir, dependiendo de la plataforma y el entorno específicos.

Claramente, es muy peligroso desarrollar un programa sin comprobar los avisos de compilación. Si hay alguna función no usada correctamente, causará que el programa falle o produzca resultados incorrectos. Activando los avisos del compilador con la opción '`-Wall`' se detectarán muchos de los más habituales errores que ocurren programando en C.

2.3 Compilando múltiples archivos fuentes

Un programa puede ser dividido en múltiples ficheros. Facilitando tanto la edición como la comprensión del código, especialmente en el caso de largos programas —también permite que las partes individuales sean compiladas de manera independiente.

En el siguiente ejemplo se dividirá el programa *Hola Mundo* en tres ficheros: '`main.es.c`', '`hola_fn.es.c`' y el fichero de cabecera '`hola.es.h`'. Aquí está el programa principal '`main.es.c`':

```
#include "hola.h"

int
main (void)
{
  hola ("mundo");
  return 0;
}
```

La llamada original a la función de sistema `printf` en el programa previo '`hola.es.c`' ha sido reemplazado por una llamada a una nueva función externa `hola`, que se definirá en un fichero separado '`hola_fn.es.c`'

El programa main también incluye el fichero de cabecera '`hola.es.h`' que contendrá la declaración de la función `hola`. La declaración es usada para asegurar que los tipos de los argumentos y el valor de retorno concuerda correctamente con la llamada de la función y la definición de la función. No se necesita incluir el fichero de cabecera '`stdio.h`' en '`main.es.c`' para declarar la función `printf`, ya que el fichero '`main.es.c`' no llama a `printf` directamente.

La declaración en '`hola.es.h`' es una simple línea que especifica el prototipo de la función `hola`

```
void hola (const char * nombre);
```

La definición de la función `hola` en sí está contenida en el fichero '`hola_fn.es.c`':

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

void
hola (const char * nombre)
{
  printf ("!Hola, %s!\n", nombre);
}
```

Esta función imprime el mensaje "¡Hola, *nombre* !" usando como valor de *nombre* el argumento introducido.

Casualmente, la diferencia entre las dos formas de la instrucción de inclusión `#include "`*FILE*`.h"` y `#include <`*FILE*`.h>` es que la primera busca el archivo '*FILE*.h' en el directorio actual antes de buscarlo en los directorios de los archivos de cabeceras del sistema. La instrucción de inclusión `#include <`*FILE*`.h>` busca por defecto los archivos de cabeceras del sistema, pero no busca en el directorio actual.

Para compilar estos ficheros fuente con `gcc`, se usa el siguiente comando:

```
$ gcc -Wall main.es.c hola_fn.es.c -o nuevohola
```

En este caso, se usa la opción '`-o`' para especificar un nombre al fichero de salida diferente para el ejecutable, '`nuevohola`'. Nótese que el fichero de cabecera '`hola.es.h`' no está especificado en la lista de ficheros en la línea de comandos. La directiva `#include`

`"hola.es.h"` en los ficheros fuente ordena al compilador incluirlo de forma automática en los puntos apropiados.

Para ejecutar el programa, se escribe la ruta del ejecutable:

```
$ ./nuevohola
¡Hola, mundo!
```

Todas las partes del programa han sido combinadas en un solo fichero ejecutable, que produce el mismo resultado que el ejecutable creado desde el fichero fuente usado anteriormente.

2.4 Compilando archivos independientes

Si un programa es almacenado en un solo fichero, entonces cualquier cambio en una función individual requiere que el programa entero sea recompilado para producir un nuevo ejecutable. La recompilación de largos ficheros fuente puede consumir mucho tiempo.

Cuando los programas son almacenados en ficheros fuente independientes, solo los ficheros que han cambiado necesitan ser recompilados después de que el código fuente haya sido modificado. De este modo, los ficheros fuente son compilados separadamente y *enlazados* juntos —es un proceso de dos fases. En la primera fase, un fichero es compilado sin crear el ejecutable. El resultado es un *fichero objeto*, y tiene la extensión '`.o`' al usar GCC.

En la segunda fase, los ficheros objeto son unidos por otro programa llamado *enlazador*. El enlazador combina todos los ficheros objeto creando un solo ejecutable.

Un fichero objeto contiene código máquina en el cual las referencias a direcciones de memoria de funciones (ó variables) de otros ficheros se dejan indefinidas. Esto permite que los ficheros fuentes se compilen sin referencia directa a otros. El enlazador rellena estas direcciones perdidas cuando produce el ejecutable.

2.4.1 Creando archivos objeto desde archivos fuente

La opción '`-c`' es usada para compilar un fichero fuente para crear un fichero objeto. Por ejemplo, el siguiente comando compilará el fichero fuente '`main.es.c`' para generar un fichero objeto:

```
$ gcc -Wall -c main.es.c
```

Esto produce un fichero objeto '`main.o`' que contiene el código máquina para la función `main`. Éste contiene una referencia la función externa `hola`, pero la correspondiente dirección de memoria se deja indefinida en el fichero objeto en esta fase (se introducirá después al enlazarse).

El correspondiente comando para compilar la función `hola` en el código fuente '`hola_fn.es.c`' es:

```
$ gcc -Wall -c hola_fn.es.c
```

Esto produce el fichero objeto '`hola_fn.o`'.

Nótese que no hay necesidad de usar la opción '`-o`' para especificar el nombre del fichero de salida en este caso. Al compilar con la opción '`-c`' el compilador de forma automática crea un fichero objeto cuyo nombre es el mismo que el fichero fuente, pero con '`.o`' en vez de la extensión original.

No hay necesidad de poner el fichero de cabecera '`hola.es.h`' en la línea de comandos, ya que se incluye de forma automática gracias a las sentencias `#include` en '`main.es.c`' y '`hola_fn.es.c`'.

2.4.2 Creando ejecutables desde archivos objeto

El paso final para crear un fichero ejecutable es usar `gcc` para enlazar los ficheros objetos juntos y llenar las direcciones perdidas de funciones externas. Para enlazar ficheros objetos juntos, simplemente se listan en el siguiente comando:

```
$ gcc main.o hola_fn.o -o hola
```

Esta es una de las pocas ocasiones en las que no hay necesidad de usar la opción de avisos '`-Wall`', debido a que los ficheros fuente individuales han sido compilados exitosamente en código objeto. Una vez que los ficheros fuente han sido compilados, enlazar es un proceso ambiguo que tendrá éxito o fallará (falla solo si hay referencias que no pueden ser resueltas).

Para realizar el paso de enlazar `gcc` usar el enlazador `ld`, que es un programa separado. En sistemas GNU se usa el enlazador de GNU, GNU `ld`. Otros sistemas pueden usar el enlazador de GNU con GCC, o pueden tener sus propios enlazadores. El enlazador en sí será discutido después (véase Capítulo 11 [Como funciona el compilador], página 107). Al ejecutar el enlazador, `gcc` crea un fichero ejecutable desde los ficheros objeto.

El fichero ejecutable resultante puede ejecutarse ahora:

```
$ ./hola
¡Hola, mundo!
```

Se produce la misma salida que en la versión del programa que usa un solo fichero fuente visto en la sección previa.

2.5 Recompilando y reenlazando

Para mostrar cómo los ficheros fuente pueden ser compilados de manera independiente se editará el programa '`main.es.c`' para imprimir un saludo para `cualquiera` en vez de `mundo`:

```
#include "hola.h"

int
main (void)
{
  hola ("cualquiera");  /* se cambia "mundo" */
  return 0;
}
```

El fichero actualizado '`main.es.c`' ahora puede ser recompilado con el siguiente comando:

```
$ gcc -Wall -c main2.es.c
```

Esto produce un nuevo fichero objeto '`main.o`'. No se necesita crear un nuevo fichero objeto para '`hola_fn.es.c`', debido a que el fichero y los

ficheros relacionados de los que depende, tales como ficheros de cabeceras, no han cambiado.

El nuevo fichero objeto puede ser reenlazado con la función `hola` para crear un nuevo fichero ejecutable:

```
$ gcc main2.es.o hola_fn.o -o hola
```

El ejecutable resultante '`hola`' ahora usa la nueva función `main` para producir la siguiente salida:

```
$ ./hola
¡Hola, cualquiera!
```

Nótese que solo el fichero '`main.es.c`' ha sido recompilado y, por tanto, reenlazado con el fichero objeto existente para la función `hola`. Si el fichero '`hola_fn.es.c`' hubiera sido modificado, se podría haber recompilado '`hola_fn.es.c`' para crear un nuevo fichero objeto '`hola_fn.o`' y reenlazar este con el fichero '`main.o`'.[1]

En un gran proyecto con muchos ficheros fuente, recompilar solo aquellos que han sido modificados crea un significativo ahorro. El proceso de recompilar solo los ficheros modificados en un proyecto puede ser automatizado con el programa estándar de Unix `make`.

[1] Si el prototipo de una función cambia, es necesario modificar y recompilar todos los ficheros fuentes que la usan.

2.6 Un pequeño makefile

Para aquellos no familiarizados con `make`, esta sección provee una demostración de su uso. `make` es un programa propio que puede ser encontrado en todos los sistemas Unix. Para aprender más acerca de la versión GNU de `make` se necesitará consultar el manual de *GNU Make* escrito por Richard M. Stallman y Roland McGrath (véase [Lectura adicional], página 131).

`make` lee una descripción de un proyecto desde un archivo conocido por *makefile* (por defecto, llamado '`Makefile`' en el directorio actual). Un makefile especifica un conjunto de reglas de compilación en términos de *objetivos* (tal como ejecutables) y sus *dependencias* (tal como ficheros objeto y ficheros fuente) en el siguiente formato:

```
objetivo: dependencias
        comando
```

Por cada objetivo, make chequea el momento de modificación de los correspondientes ficheros de dependencia para determinar si el objetivo necesita ser reconstruido usando el correspondiente comando. Nótese que las líneas de *`comandos`* en un makefile deben ser indentadas con un carácter `TAB`, sin espacios.

GNU Make contiene muchas reglas por defecto, llamadas reglas *implícitas*, para simplificar la construcción de makefiles. Por ejemplo, estos especifican que ficheros '`.o`' pueden ser obtenidos desde ficheros '`.c`' al compilarse, y que un ejecutable puede ser creado enlazando ficheros '`.o`' juntos. Las reglas implícitas son definidas en términos de *variables make*, tales como `CC` (el compilador de C) y `CFLAGS` (las opciones de compilación para programas C), que pueden ser asignadas usando líneas *`VARIABLE=VALUE`* en el makefile. Para C++ las variables equivalentes son `CXX` y `CXXFLAGS`, mientras la variable `CPPFLAGS` asigna las opciones del preprocesador. Las reglas implícitas y las definidas por el usuario se encadenadan juntas de forma automática como GNU Make necesite.

Un '`Makefile`' simple para el proyecto anterior puede ser escrito como sigue:

```
CC=gcc
CFLAGS=-Wall
main:  main.o hello_fn.o

clean:
        rm -f main main.o hello_fn.o
```

El fichero puede ser leido de la manera siguiente: usando el compilador de C `gcc`, con la opción de compilación '`-Wall`', se construirá el objetivo ejecutable `main` desde los ficheros objeto '`main.o`'

y '`hola_fn.o`' (estos, en cambio, serán construidos vía reglas implícitas desde '`main.es.c`' y '`hola_fn.es.c`'. El objetivo `clean` no tiene dependencias y simplemente elimina todos los ficheros compilados.[2] La opción '`-f`' (fuerza) que en el comando `rm` se suprima cualquier mensaje de error si los ficheros no existen.

Para usar el makefile, se escribe `make`. Al llamarse sin argumentos, se construye el primer objetivo en el makefile, produciendo el ejecutable '`main`'.

```
$ make
gcc -Wall -c -o main.o main.es.c
gcc -Wall -c -o hola_fn.o hola_fn.es.c
gcc main.o hola_fn.o -o main
$ ./main
¡Hola, mundo!
```

Para reconstruir el ejecutable después de modificar un fichero fuente, simplemente se escribe `make` de nuevo. Al comprobar las fechas de modificación de los objetivos y de los ficheros dependientes, make identifica los ficheros que han cambiado y regenera los ficheros intermedios correspondientes y necesarios para actualizar los objetivos:

```
$ emacs main.es.c (editar el fichero)
$ make
gcc -Wall -c -o main.o main.es.c
gcc main.o hola_fn.o -o main
$ ./main
¡Hola, cualquiera!
```

Finalmente, para eliminar los ficheros generados, se escribe `make clean`:

```
$ make clean
rm -f main main.o hola_fn.o
```

Un makefile más sofisticado normalmente contiene objetivos adicionales para instalación (`make install`) y testeo (`make check`).

Los ejemplos del resto de este libro son suficientemente pequeños para no necesitar makefiles, pero el uso de make se recomienda en programas largos.

---

[2] Esto asume que no hay ningún fichero llamado '`clean`' en el directorio actual —ver la discusión de 'falsos objetivos' en el manual GNU Make para más detalles.

2.7 Enlazando con librerías externas

Una librería es una colección de ficheros objetos precompilados que pueden ser enlazados dentro de programas. El uso más común de librerías es proporcionar funciones de sistema, tales como la función raíz cuadrada `sqrt` que se encuentra en la librería matemática de C.

Las librerías suelen almacenarse en *ficheros de archivo* especiales con la extensión '`.a`', también llamadas *librerías estáticas*. Éstas son creadas desde ficheros objeto con una herramienta propia, el archivador de GNU `ar`, y es usado por el enlazador para resolver referencias a funciones en tiempo de compilación. Después veremos cómo crear librerías usando el comando `ar` (véase Capítulo 10 [Utilidades relativas al compilador], página 99). Por simplicidad, sólo las librerías estáticas se cubren en esta sección —el enlazado dinámico en tiempo de ejecución usando *librerías compartidas* será descrito en el siguiente capítulo.

El sistema de librerías estándar normalmente se encuentra en los directorios '`/usr/lib`' y '`/lib`'.[3] Las librerías algunas veces pueden ser encontradas en un directorio específico de la arquitectura, como '`/usr/lib/i386-linux-gnu/`'. Por ejemplo, la librería matemática de C está almacenada normalmente en el fichero '`/usr/lib/libm.a`' en sistemas tipo Unix. Las declaraciones de prototipo correspondientes a las funciones de esta librería se realizan en el fichero de cabecera '`/usr/include/math.h`'. La librería estándar de C en sí misma es almacenada en '`/usr/lib/libc.a`' y contiene funciones especificadas en el estándar ANSI/ISO C, tal como '`printf`' —esta librería es enlazada por defecto en cada programa C.

Aquí se muestra un programa de ejemplo que realiza una llamada a la función externa `sqrt` en la librería matemática '`libm.a`':

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

int
main (void)
{
  double x = 2.0;
  double y = sqrt (x);
  printf ("La raiz cuadrada de %f es %f\n", x, y);
```

[3] En sistemas que soportan versiones de librerías ejecutables de 32 y 64 bits, las versiones de 64 bit se almacenarán frecuentemente en '`/usr/lib64`' y '`/lib64`', y las versiones de 32 bits en '`/usr/lib`' y '`lib`'.

```
  return 0;
}
```

Intentar crear un ejecutable desde este único fichero fuente causa que el compilador devuelva un error en la fase de enlace:

```
$ gcc -Wall calc.es.c -o calc
/tmp/ccbR60jm.o: In function ‘main’:
/tmp/ccbR60jm.o(.text+0x19): undefined reference
  to ‘sqrt’
```

El problema es que la referencia a la función `sqrt` no puede ser resuelta sin la librería matemática externa '`libm.a`'. La función `sqrt` no está definida en el programa o en la librería por defecto '`libc.a`', y el compilador no enlaza al fichero '`libm.a`' a menos que éste esté explícitamente seleccionado. Casualmente, el fichero mencionado el mensaje de error '`/tmp/ccbR60jm.o`' es un fichero objeto temporal creado por el compilador desde '`calc.es.c`', para llevar a cabo el proceso de enlace.

Para permitir que el compilador enlace la función `sqrt` al programa main de '`calc.es.c`' es necesario proveer la librería '`libm.a`'. Un obvio pero no convencional modo de hacer esto es especificarlo de manera explícita en la línea de comandos:

```
$ gcc -Wall calc.es.c /usr/lib/libm.a -o calc
```

La librería '`libm.a`' contiene los ficheros objeto para todas las funciones matemáticas, tales como `sin`, `cos`, `exp`, `log` y `sqrt`. El enlazador busca a través de estos para encontrar el fichero objeto conteniendo la función `sqrt`.

Una vez que el fichero objeto para la función `sqrt` fué encontrado, el programa main puede ser enlazado y se produce un ejecutable completo:

```
$ ./calc
La raíz cuadrada de 2.000000 es 1.414214
```

El fichero ejecutable incluye el código máquina para la función main y el código máquina para la función `sqrt`, copiado desde el archivo objeto correspondiente en la librería '`libm.a`'.

Para evitar la necesidad de especificar largas rutas en la línea de comandos, el compilador proporciona una opción de atajo '`-l`' para enlazar librerías. Por ejemplo, el siguiente comando,

```
$ gcc -Wall calc.es.c -lm -o calc
```

es equivalente al comando original usando el nombre de la librería completa '`/usr/lib/libm.a`'.

En general, la opción del compilador '`-l`*NAME*' intentará enlazar ficheros objeto con un fichero de librería '`lib`*NAME*`.a`' en los directorios de librería estándar. Los directorios adicionales pueden especificarse con opciones de la línea comandos y variables de entorno, por decirlo resumidamente. Un programa largo normalmente usará muchas opciones '`-l`' para enlazar a librerías tales como la librería matemática, librerías gráficas y librerías de red.

2.7.1 Orden de enlace de librerías

El comportamiento tradicional de los enlazadores es buscar funciones externas de izquierda a derecha en las librerías especificadas en la línea de comandos. Esto significa que una librería que contiene la definición de una función debería aparecer después de cualquier fichero fuente o ficheros objeto que la usen. Esto incluye librerías especificadas con la opción '`-l`', como se muestra en el siguiente comando:

```
$ gcc -Wall calc.es.c -lm -o calc    (orden correcto)
```

Con algunos enlazadores el orden opuesto (poner la opción '`-lm`' antes del fichero que lo usa) daría como resultado un error,

```
$ cc -Wall -lm calc.es.c -o calc    (orden incorrecto)
main.o: In function ‘main’:
main.o(.text+0xf): undefined reference to ‘sqrt’
```

debido a que no hay librería o fichero objeto que contenga `sqrt` después de '`calc.es.c`'. La opción '`-lm`'

debería aparecer después del fichero '`calc.es.c`'.

Cuando se usan varias librerías, se debería usar la misma convención que con las librerías en sí. Una librería que llama a una función externa definida en otra librería debería aparecer antes que la librería que contiene la función.

Por ejemplo, un programa '`data.es.c`' usando la librería GNU Linear Programming library '`libglpk.a`', que a su vez usa la librería matemática '`libm.a`', se compila así:

```
$ gcc -Wall data.es.c -lglpk -lm
```

debido a que los fichero objetos de '`libglpk.a`' usan funciones definidas en '`libm.a`'.

La mayoría de los enlazadores actuales buscarán todas las librerías, sin importar el orden, pero debido a que algunos no hacen esto es mejor seguir la convención de ordenar las librerías de izquierda a derecha.

Es útil tener todo esto en mente si se encuentran problemas inesperados con referencias no definidas, y mirar si todas las librerías necesarias aparecen en la línea de comandos.

2.8 Usando librerías de archivos de cabeceras

Cuando se usa una librería es esencial incluir los ficheros de cabecera apropiados, para así declarar los argumentos de funciones y devolver valores con los tipos correctos. Sin declaraciones, los argumentos de una función pueden pasar el tipo erróneo, causando resultados corruptos.

El siguiente ejemplo muestra otro programa que crea una llamada de función para la librería estándar de C. En este caso, la función `strtod` es usada para convertir una cadena `\"123\"` a un número en coma flotante:

```
#include <stdio.h>

int
main (void)
{
  double x = strtod ("123", NULL);
  printf ("El valor es %f\n", x);
  return 0;
}
```

Sin embargo, el programa contiene un error —la sentencia `#include` para la necesaria cabecera '`stdlib.h`' no se encuentra, así el prototipo `double strtod (const char * string, char * tail)` no será visto por el compilador.

La compilación del programa sin opciones de aviso producirá un fichero ejecutable que da resultados incorrectos:

```
$ gcc badconv.es.c -lm
$ ./a.out
El valor es 966656.000000 (resultado incorrecto,
debería ser 123.0)
```

Los resultados están corruptos porque los argumento y el valor devuelto de la llamada a `strtod` son pasados con tipos incorrectos.[4] Esto puede ser detectado activando la opción de avisos '`-Wall`':

[4] La actual salida anterior puede diferir, dependiendo de la plataforma y el entorno específicos.

```
$ gcc -Wall badconv.es.c -lm
badconv.es.c: In function ‘main’:
badconv.es.c:6: warning: implicit declaration of
  function ‘strtod’
```

Este ejemplo muestra de nuevo la importancia de usar la opción de aviso ‘`-Wall`’ para detectar problemas serios que de otro modo no podrían ser fácilmente localizados.

3 Opciones de compilación

Este capítulo describe otras opciones del compilador comúnmente usadas disponibles en GCC. Estas opciones controlan funcionalidades tales como buscar rutas usadas para localizar librerías e incluir ficheros, el uso de avisos adicionales y diagnósticos, macros del preprocesador y dialectos del lenguaje C.

3.1 Asignando rutas de búsqueda

En el último capítulo, se vió cómo enlazar a un programa con funciones de la librería matemática '`libm.a`', usando la opción de atajo '`-lm`' y el fichero de cabecera '`math.h`'.

Un problema común cuando se compila un programa usando ficheros de cabecera es el error:

```
FILE.h: No such file or directory
```

Esto ocurre si un fichero de cabecera no está presente en los directorios de ficheros include estándar usados por `gcc`. Un problema similar puede ocurrir en librerías:

```
/usr/bin/ld: cannot find library
```

Esto ocurre si una librería usada para enlazar no está presente en los directorios de librería estándar usados por `gcc`.

Por defecto, `gcc` busca en los siguientes directorios los ficheros de cabecera:

```
/usr/local/include/
/usr/include/
```

y los siguientes directorios para librerías:

```
/usr/local/lib/
/usr/lib/
```

La lista de directorios para ficheros de cabecera son referidos con frecuencia como *ruta de include*, y la lista de directorios para librerías como *ruta de búsqueda de librerías* o *ruta de enlace*.

Los directorios en estas rutas son buscados en orden, desde el primero hasta el último en los dos listados de arriba.[1] Por ejemplo, un fichero de cabecera encontrado en '`/usr/local/include`' precede a un fichero con el mismo nombre en '`/usr/include`'. De

[1] Las rutas de búsqueda por defecto pueden también incluir directorios adicionales y dependientes del sistema o específicos del sitio, y directorios en la instalación de GCC en sí. Por ejemplo, en plataformas de 64 bits directorios adicionales '`lib64`' pueden también ser búscados por defecto.

manera similar, una librería encontrada en '`/usr/local/lib`' precederá a una librería con el mismo nombre en '`/usr/lib`'.

Cuando se instalan librerías adicionales en otros directorios es necesario extender las rutas de búsqueda, para que las librerías sean encontradas. Las opciones del compilador '`-I`' y '`-L`' añaden nuevos directorios al principio de la ruta include y la ruta de búsqueda de librería respectivamente.

3.1.1 Ejemplo de ruta de búsqueda

El siguiente programa de ejemplo usa una librería que podría ser instalada como un paquete adicional en un sistema —la GNU Database Management Library (GDBM). La librería GDBM almacena pares clave:valor en un fichero DBM, un tipo de fichero de datos que permite que los valores sean almacenados e indexados por una *clave* (una arbitraria secuencia de caracteres). Aquí está el programa de ejemplo '`dbmain.es.c`', que crea un fichero DBM conteniendo una clave '`clavetest`' con el valor '`valortest`':

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

int
main (void)
{
  GDBM_FILE dbf;
  datum key = { "clavetest", 7 };   /* clave, tamano */
  datum value = { "valortest", 9 }; /* valor, tamano */

  printf ("Almacenado el par clave-valor...  ");
  dbf = gdbm_open ("test", 0, GDBM_NEWDB, 0644, 0);
  gdbm_store (dbf, key, value, GDBM_INSERT);
  gdbm_close (dbf);
  printf ("hecho.\n");
  return 0;
}
```

El programa usa el fichero de cabecera '`gdbm.h`' y la librería '`libgdbm.a`'. Si la librería ha sido instalada en la localización por defecto '`/usr/local/lib`', con el fichero de cabecera en '`/usr/local/include`', entonces el programa puede ser compilado con el siguiente comando:

```
$ gcc -Wall dbmain.es.c -lgdbm
```

Ambos de estos directorios son parte del `gcc` por defecto e incluyen enlaces y rutas.

Sin embargo, si GDBM ha sido instalado en una localización diferente, al intentar compilar el programa se obtendrá el siguiente error:

```
$ gcc -Wall dbmain.es.c -lgdbm
dbmain.es.c:1: gdbm.h: No such file or directory
```

Por ejemplo, si la versión 1.8.3 del paquete GDBM está instalada en el directorio '`/opt/gdbm-1.8.3`' la localización del fichero de cabecera sería,

```
/opt/gdbm-1.8.3/include/gdbm.h
```

que no es parte de la ruta include del `gcc` por defecto. Añadiendo el directorio apropiado para la ruta include con la opción '`-I`' en línea de comandos, permite que el programa se compile, pero no se enlace:

```
$ gcc -Wall -I/opt/gdbm-1.8.3/include dbmain.es.c -lgdbm
/usr/bin/ld: cannot find -lgdbm
collect2: ld returned 1 exit status
```

El directorio que contiene la librería no está incluido en la ruta de enlace. Éste puede ser añadido a la ruta de enlace usando la siguiente opción:

```
-L/opt/gdbm-1.8.3/lib/
```

El siguiente comando permite que el programa sea compilado y enlazado:

```
$ gcc -Wall -I/opt/gdbm-1.8.3/include
   -L/opt/gdbm-1.8.3/lib dbmain.es.c -lgdbm
```

Esto produce el ejecutable final enlazado a la librería GDBM. Antes de ver como funciona este ejecutable se dará una breve mirada a las variables de entorno que afecta a las opciones '`-I`' y '`-L`'.

Nótese que nunca se localizarán las rutas absolutas de los ficheros de cabecera en sentencias `#include` en el código fuente, ya que esto evitaría que el programa se compilase en otros sistemas. La opción '`-I`' o la variable `INCLUDE_PATH` descrita anteriormente siempre se debería usar para asignar la ruta include para los ficheros de cabecera.

3.1.2 Variables de entorno

La búsqueda de rutas para los ficheros de cabecera y librerías puede también ser controlado a través de las variables de entorno en la shell. Estas pueden ser asignadas de manera automática por cada sesión usando el apropiado fichero de acceso, tal como '`.bash_profile`' en el caso de GNU Bash.

Los directorios adicionales pueden ser añadidos para la ruta include usando la variable de entorno `C_INCLUDE_PATH` (para ficheros de cabecera de C). Por ejemplo, los siguientes comandos añadirán '`/opt/gdbm-1.8.3/include`' a la ruta include cuando se compilan programas C:

```
$ C_INCLUDE_PATH=/opt/gdbm-1.8.3/include
$ export C_INCLUDE_PATH
```

y de manera similar para programas C++:

```
$ CPLUS_INCLUDE_PATH=/opt/gdbm-1.8.3/include
$ export CPLUS_INCLUDE_PATH
```

Este directorio será buscado después de cualquier directorio especificado en el comando con la opción '`-I`', y antes de los directorios estándar por defecto (tales como '`/usr/local/include`' y '`/usr/include`'). El comando de shell `export` es necesario para crear la variable de entorno disponible a programas fuera de la shell en sí, tales como el compilador —es solo necesario unavez por cada variable en cada sesión de shell, y también puede ser asignado en el apropiado fichero de login.[2]

De manera similar, directorios adicionales pueden ser añadidos a la ruta de enlace usando la variable de entorno `LIBRARY_PATH`. Por ejemplo, los siguientes comandos añadirán '`/opt/gdbm-1.8.3/lib`' a la ruta de enlace.

```
$ LIBRARY_PATH=/opt/gdbm-1.8.3/lib
$ export LIBRARY_PATH
```

Este directorio será buscado después de que cualquier directorio especificado en la línea de comandos con la opción '`-L`', y antes que los directorios estándar por defecto (tales como '`/usr/local/lib`' y '`/usr/lib`').

Con las configuraciones de las variables de entorno dadas arriba el programa '`dbmain.es.c`' puede ser compilado sin las opciones '`-I`'

y '`-L`',

[2] En GNU Bash, la forma corta `export` *`VARIABLE=VALOR`* también se permite.

```
$ gcc -Wall dbmain.es.c -lgdbm
```

porque la rutas por defecto ahora usan los directorios especificados en las variables de entorno `C_INCLUDE_PATH` y `LIBRARY_PATH`. El mismo comando de compilación con `g++` debería usar las variables de entorno `CPLUS_INCLUDE_PATH` y `LIBRARY_PATH`.

3.1.3 Rutas de búsqueda extendidas

Siguiendo la convención estándar de Unix para la búsqueda de rutas, varios directorios pueden ser especificados en una variable de entorno como una lista cuyos elementos se separan con el carácter dos puntos ':':

DIR1:*DIR2*:*DIR3*:...

Los directorios son buscados de izquierda a derecha. Un punto simple '`.`' puede ser usado para especificar el directorio actual.[3]

Por ejemplo, la siguiente configuración crea un include por defecto y enlaza rutas para paquetes instalados en el directorio actual '`.`' y en los directorios '`include`' y '`lib`' bajo '`/opt/gdbm-1.8.3`' y '`/net`' respectivamente:

```
$ C_INCLUDE_PATH=.:/opt/gdbm-1.8.3/include:/net/include
$ LIBRARY_PATH=.:/opt/gdbm-1.8.3/lib:/net/lib
```

Para programas C++, se usa la variable de entorno `CPLUS_INCLUDE_PATH` en vez de `C_INCLUDE_PATH`.

Para especificar múltiples rutas de búsqueda en la línea de comandos, las opciones '`-I`' y '`-L`' pueden ser repetidas. Por ejemplo, el siguiente comando,

```
$ gcc -I. -I/opt/gdbm-1.8.3/include -I/net/include
   -L. -L/opt/gdbm-1.8.3/lib -L/net/lib .....
```

es equivalente a las configuraciones en la variable de entornos dadas anteriormente.

Cuando las variables de entorno y las opciones de comandos son usados juntos el compilador busca los directorios en el siguiente orden:

1. opciones de comandos '`-I`' y '`-L`', de izquierda a derecha
2. directorios especificados por variables de entornos, tales como `C_INCLUDE_PATH` (para programas C), `CPLUS_INCLUDE_PATH` (para programas C++) y `LIBRARY_PATH`
3. directorios de sistema por defecto

[3] El directorio actual también puede ser especificado usando una ruta vacía. Por ejemplo, :*DIR1*:*DIR2* es equivalente a .:*DIR1*:*DIR2*.

En el uso del día a día, los directorios son normalmente añadidos a rutas de búsqueda con las opciones '`-I`' y '`-L`'.

3.2 Librerías compartidas y librerías estáticas

Aunque el programa de ejemplo de arriba ha sido exitosamente compilado y enlazado, es necesario un paso final antes de ser capaz de cargar y ejecutar el fichero ejecutable.

En un intento por iniciar la ejecución directamente, sucederá el siguiente error en la mayoría de los sistemas:

```
$ ./a.out
./a.out: error while loading shared libraries:
libgdbm.so.3: cannot open shared object file:
No such file or directory
```

Esto es porque el paquete GDBM proporciona una *librería compartida*. Este tipo de librería requiere un tratamiento especial —debe ser cargada desde disco antes de que el ejecutable cargue.

Las librerías externas se proveen normalmente de dos formas: *librerías estáticas* y *librerías compartidas*. Las librerías estáticas son los ficheros '`.a`' vistos antes. Cuando un programa es enlazado a una librería estática, el código máquina de los ficheros objetos para cualquier función externa usada por el programa es copiado desde la librería al ejecutable final.

Las librerías compartidas son manejadas con una forma más avanzada de enlace, que crea el fichero ejecutable más pequeño. Ellas usan la extensión '`.so`', soportado por los *objetos compartidos*.

Un fichero ejecutable enlazado a una librería compartida contiene solo una pequeña parte de las funciones requeridas, en vez del código máquina completo de los ficheros para las funciones externas. Antes de que el fichero ejecutable empiece su ejecución, el código máquina de las funciones externas es copiado en memoria desde el fichero de la librería compartida por el sistema operativo —un proceso referido como *enlace dinámico*.

El enlace dinámico crea ficheros ejecutables pequeños y reserva espacio en disco, porque una copia de una librería puede ser compartida entre múltiples programas. En la mayoría de los sistemas operativos también se proporciona un mecanismo de memoria virtual que permite que una copia de una librería compartida en memoria física sea usada por todos los programas en ejecución, ahorrando tanto memoria como espacio en disco.

Además, las librerías compartidas hacen posible actualizar una librería sin recompilar los programas que ella usa (la interfaz dada a la librería no cambia).

Por estas ventajas `gcc` compila programas para usar librerías compartidas por defecto en la mayoría de los sistemas, si éstas están disponibles. Siempre y cuando una librería estática '`lib`*`NAME`*`.a`' sea usada para enlazarse con la opción '`-l`*`NAME`*' el compilador primero busca una librería compartida alternativa con el mismo nombre y una extensión '`.so`'.

En este caso, cuando el compilador busca la librería '`libgdbm`' en la ruta de enlace, éste encuentra los dos ficheros siguientes en el directorio '`/opt/gdbm-1.8.3/lib`':

```
$ cd /opt/gdbm-1.8.3/lib
$ ls libgdbm.*
libgdbm.a  libgdbm.so
```

Por consiguiente, el fichero del objeto compartido '`libgdbm.so`' es usado de manera preferente a la librería estática '`libgdbm.a`'.

Sin embargo, cuando el fichero ejecutable es iniciado su función de carga, debe encontrar la librería compartida para cargarlo en memoria. Por defecto el cargador busca librerías compartidas sólo en un conjunto predefinido de directorios de sistema tales como '`/usr/local/lib`' y '`/usr/lib`'. Si la librería no está localizada en uno de estos directorios debe ser añadida a la ruta de carga.[4]

El camino más simple para definir la ruta de carga es a través de la variable de entorno `LD_LIBRARY_PATH`. Por ejemplo, los comandos siguientes definen la ruta de carga a '`/opt/gdbm-1.8.3/lib`' y así el fichero '`libgdbm.so`' puede ser encontrado:

```
$ LD_LIBRARY_PATH=/opt/gdbm-1.8.3/lib
$ export LD_LIBRARY_PATH
$ ./a.out
Almacenado el par clave-valor... hecho.
```

El ejecutable ahora funciona bien, imprime su mensaje y crea un fichero DBM llamado '`test`' conteniendo el par clave:valor '`clavetest`' y '`valortest`'.

Para ahorrar escritura, la variable de entorno `LD_LIBRARY_PATH` puede ser definida de forma automática por cada sesión usando el

[4] Nótese que el directorio que contiene la librería compartida puede, en principio, ser guardado ("hard-coded") en el propio ejecutable usando la opción del enlazador '`-rpath`', pero esto no es hecho normalmente debido a que da problemas si la librería se ha movido o el ejecutable está copiado a otro sistema.

apropiado fichero de login, tal como '`.bash_profile`' para la shell GNU Bash.

Varios directorios de librerías compartidas pueden ser fijadas en la ruta de carga, como una lista separada por dos puntos *DIR1*:*DIR2*:*DIR3*:...:*DIRN*. Por ejemplo, el siguiente comando define la ruta de carga para usar los directorios '`lib`' bajo '`/opt/gdbm-1.8.3`' y '`/opt/gtk-1.4`':

```
$ LD_LIBRARY_PATH=/opt/gdbm-1.8.3/lib:/opt/gtk-1.4/lib
$ export LD_LIBRARY_PATH
```

Si la ruta de carga contiene entradas existentes, puede ser extendida usando la sintaxis `LD_LIBRARY_PATH=`*NEWDIRS*`:$LD_LIBRARY_PATH`. Por ejemplo, el siguiente comando añade el directorio '`/opt/gsl-1.5/lib`' a la ruta mostrada arriba:

```
$ LD_LIBRARY_PATH=/opt/gsl-1.5/lib:$LD_LIBRARY_PATH
$ echo $LD_LIBRARY_PATH
/opt/gsl-1.5/lib:/opt/gdbm-1.8.3/lib:/opt/gtk-1.4/lib
```

Para el administrador de sistemas es posible definir la variables `LD_LIBRARY_PATH` para todos los usuarios, añadiéndola al script de login por defecto, normalmente '`/etc/profile`'. En sistemas GNU, una ruta de sistema puede ser también definida en el fichero de configuración del cargador '`/etc/ld.so.conf`'.

De manera alternativa, el enlace estático puede ser forzado con la opción '`-static`' de `gcc` para evitar el uso de librerías compartidas:

```
$ gcc -Wall -static -I/opt/gdbm-1.8.3/include/
    -L/opt/gdbm-1.8.3/lib/ dbmain.es.c -lgdbm
```

Esto crea un ejecutable enlazado con la librería estática '`libgdbm.a`' que puede ser ejecutada sin asignar la variable de entorno `LD_LIBRARY_PATH` o poniendo librerías compartidas en los directorios por defecto:

```
$ ./a.out
Almacenando el par clave-valor... hecho.
```

Como ya se comentó, también es posible enlazar directamente con librerías de ficheros individuales especificando la ruta completa a la librería en la línea de comandos. Por ejemplo, el siguiente comando enlazará directamente con la librería estática '`libgdbm.a`',

```
$ gcc -Wall -I/opt/gdbm-1.8.3/include
    dbmain.es.c /opt/gdbm-1.8.3/lib/libgdbm.a
```

y el comando de abajo enlazará con el fichero de librería compartida '`libgdbm.so`':

```
$ gcc -Wall -I/opt/gdbm-1.8.3/include
    dbmain.es.c /opt/gdbm-1.8.3/lib/libgdbm.so
```

En el último caso aún es necesario definir la ruta de carga de librería cuando se ejecuta el ejecutable.

3.3 Estándares del lenguaje C

Por defecto, `gcc` compila programas usando el dialecto GNU del lenguaje C, llamado como *GNU C*. Este dialecto incorpora el estándar oficial ANSI/ISO para el lenguaje C con varias extensiones útiles de GNU, tales como funciones anidadas y vectores de tamaño variable. La mayoría de los programas ANSI/ISO compilarán bajo GNU C sin cambios.

Hay varias opciones que controlan el dialecto de C usado por `gcc`. Las opciones más comúnmente usadas son '`-ansi`' y '`-pedantic`'. Los dialectos específicos del lenguaje C por cada estándar pueden también ser seleccionadas con la opción '`-std`'.

3.3.1 ANSI/ISO

A veces un programa ANSI/ISO puede ser incompatible con las extensiones de GNU C. Para tratar con esta situación, la opción del compilador '`-ansi`' inhabilita las extensiones de que están en conflicto con el estándar ANSI/ISO. Los sistemas que usan la GNU C Library (`glibc`) también deshabilitan las extensiones a la librería estándar de C. Esto permite que los programas escritos para ANSI/ISO sean compilados sin efectos no deseados de extensiones de GNU.

Por ejemplo, aquí hay un programa ANSI/ISO C válido que usa una variable llamada `asm`:

```
#include <stdio.h>

int
main (void)
{
  const char asm[] = "6502";
  printf ("la cadena asm es '%s'\n", asm);
  return 0;
}
```

El nombre de variable `asm` es válido bajo el estándar ANSI/ISO, pero este programa no compilará en GNU C porque `asm` es una palabra reservada de la extensión GNU C (permite instrucciones

de ensamblador nativas para ser usadas en funciones C). Por consiguiente, no puede ser usado como un nombre de variable sin dar un error de compilación:

```
$ gcc -Wall ansi.c
ansi.c: In function 'main':
ansi.c:6:14: error: expected identifier or '(' before 'asm'
ansi.c:7:39: error: expected expression before 'asm'
```

En contraste, usar la opción '`-ansi`' inhabilita la palabra clave `asm`, y permite que el programa anterior sea compilado correctamente:

```
$ gcc -Wall -ansi ansi.es.c
$ ./a.out
la cadena asm es '6502'
```

Por referencia, las palabras reservadas no estándar y las macros definidas por las extensiones GNU C son `asm`, `inline`, `typeof`, `unix` y `vax`. Se pueden encontrar más detalles en el Manual de Referencia de GCC "*Using GCC*" (véase [Lectura adicional], página 131).

El siguiente ejemplo muestra el efecto de la opción '`-ansi`' en sistemas usando la GNU C Library, tales como sistemas GNU/Linux. El programa suguiente imprime el valor de pi, $\pi = 3.14159...$, dado por la definición del preprocesador `M_PI` en el fichero de cabecera '`math.h`':

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

int
main (void)
{
  printf ("el valor de pi es %f\n", M_PI);
  return 0;
}
```

La constante `M_PI` no es parte de la librería estándar ANSI/ISO de C (viene de la versión BSD de Unix). En este caso, el programa no compilará con la opción '`-ansi`':

```
$ gcc -Wall -ansi pi.c
pi.c: In function 'main':
pi.c:7:38: error: 'M_PI' undeclared (first use in this
 function)
pi.c:7:38: note: each undeclared identifier is reported
only once for each function it appears in
```

El programa puede ser compilado sin la opción '`-ansi`'. En este caso tanto el lenguaje como las extensiones de librería están habilitadas por defecto:

```
$ gcc -Wall pi.es.c
$ ./a.out
el valor de pi es 3.141593
```

También es posible compilar el programa usando ANSI/ISO C, habilitando sólo las extensiones en la GNU C Library. Esto puede ser logrado definiendo macros especiales, tales como `_GNU_SOURCE`, que habilita extensiones en GNU C Library:[5]

```
$ gcc -Wall -ansi -D_GNU_SOURCE pi.es.c
$ ./a.out
el valor de pi es 3.141593
```

La GNU C Library (Librería GNU C) proporciona un número de estas macros (referidas como *macros de test de funcionalidad*) que permiten controlar a través del soporte para extensiones POSIX (`_POSIX_C_SOURCE`), extensiones BSD (`_BSD_SOURCE`), extensiones SVID (`_SVID_SOURCE`), extensiones XOPEN (`_XOPEN_SOURCE`) y extensiones GNU (`_GNU_SOURCE`).

La macro `_GNU_SOURCE` habilita todas las extensiones juntas, siendo las extensiones precedentes al resto en los casos donde haya conflicto. Información adicional acerca de las macros de test de funcionalidad puede ser encontrada en el *GNU C Library Reference Manual* (véase [Lectura adicional], página 131).

3.3.2 ANSI/ISO estricto

La opción '`-pedantic`' del comando `gcc` en combinación con la opción '`-ansi`' hará que `gcc` rechace todas las extensiones de GNU C, no sólo aquellas que son incompatibles con el estándar ANSI/ISO. Esto ayudará a escribir programas portables que siguen el estándar ANSI/ISO.

Aquí hay un programa que usa arrays de tamaño variables, una extensión de GNU C. El array `x[n]` es declarado con un tamaño especificado por la variable entera `n`.

```
int
main (int argc, char *argv[])
{
  int i, n = argc;
  double x[n];

  for (i = 0; i < n; i++)
    x[i] = i;
```

[5] La opción '`-D`' para la definición de macros será explicada en detalle en el siguiente capítulo.

```
  return 0;
}
```

Este programa compilará con la opción '`-ansi`', porque soporta arrays de tamaño variable que no interfieren con la compilación de programas ANSI/ISO válidos —ésta es una extensión compatible hacia atrás:

```
$ gcc -Wall -ansi gnuarray.es.c
```

Sin embargo, compilar con '`-ansi -pedantic`' devuelve avisos de las violaciones del estándar ANSI/ISO:

```
$ gcc -Wall -ansi -pedantic gnuarray.es.c
gnuarray.es.c: In function ‘main’:
gnuarray.es.c:5: warning: ISO C90 forbids variable-size
  array ‘x’
```

Nótese que una ausencia de avisos con '`-ansi -pedantic`' no garantiza que un programa cumpla estrictamente con el estándar ANSI/ISO. El estándar en sí especifica sólo un limitado conjunto de circunstancias que deberían generar diagnósticos, y estos son los que se informa con '`-ansi -pedantic`'.

3.3.3 Seleccionando estándares específicos

El estándar específico del lenguaje usado por GCC puede ser controlado con la opción '`-std`'. Los siguientes estándares del lenguaje C son soportados:

'`-std=c89`' o '`-std=iso9899:1990`'
: El estándar original ANSI/ISO C (ANSI X3.159-1989, ISO/IEC 9899:1990). GCC incorpora las correcciones en los dos ISO Technical Corrigiendo al estándar original.

'`-std=iso9899:199409`'
: El lenguaje estándar ISO de C con el ISO Amendment 1, publicado en 1994. Esta enmienda era concerniente principalmente con internacionalización, tales como añadir soporte para caracteres multi-byte a la librería C.

'`-std=c99`' o '`-std=iso9899:1999`'
: El lenguaje estándar ISO de C, publicado en 1999 (ISO/IEC 9899:1999).

El lenguaje estándar de C con las extensiones de GNU puede ser seleccionado con las opciones '`-std=gnu89`' y '`-std=gnu99`'.

3.4 Opciones de aviso en `-Wall`

Tal como se describió antes (véase Sección 2.1 [Compilando un pequeño programa C], página 9), la opción de aviso '`-Wall`' habilita avisos para muchos errores comunes, y siempre se debería usar. '`-Wall`' puede combinarse con un largo número de otras opciones de aviso, más específicas, que también pueden ser seleccionadas individualmente. Aquí hay un resumen de estas opciones:

'`-Wcomment`' (incluida en '`-Wall`')
: Esta opción avisa acerca de comentarios anidados. Los comentarios anidados normalmente surgen cuando una sección de código contiene comentarios después de haber sido ya comentada:

```
/* comienzo de comentario
double x = 1.23 ; /* posición x */
*/
```

Los comentarios anidados pueden ser una fuente de confusión —el camino seguro de "comentar" una sección de código que contiene comentarios es usar la directiva del preprocesador `#if 0 ... #endif` alrededor de ella:

```
/* comentario exterior */
#if 0
double x = 1.23 ; /* posición x */
#endif
```

'`-Wformat`' (incluida en '`-Wall`')
: Esta opción avisa acerca del incorrecto uso del formato de cadenas en funciones tales como `printf` y `scanf`, donde el especificador de formato no concuerda con el tipo del correspondiente argumento de la función.

'`-Wunused`' (incluida en '`-Wall`')
: Esta opción avisa acerca de variables no usadas. Cuando una variable es declarada pero no se usa, puede ser debido a que otra variable ha sido accidentalmente sustituida en su lugar. Si la variable realmente no se necesita, ésta pueder ser eliminada del código fuente.

'`-Wimplicit`' (incluida en '`-Wall`')
: Esta opción avisa de cualquier función que es usada sin haber sido declarada. La razón más común para que una función sea usada sin haber sido declarada es haber olvidado incluir un fichero de cabecera.

‘`-Wreturn-type`’ (incluida en ‘`-Wall`’)
: Esta opción avisa de funciones que por su definición no tienen tipo de retorno, pero no son declaradas como `void`. También avisa de sentencias con un `return` vacío en funciones que no son declaradas `void`.

  Por ejemplo, el siguiente programa no usa un valor de retorno explícito:

```
#include <stdio.h>

int
main (void)
{
  printf ("hola mundo\n");
  return;
}
```

  La falta de un valor de retorno en el código de arriba podría ser el resultado de una omisión accidental por el programador —el valor devuelto por la función main es el valor de retorno de la función `printf` (el número de caracteres impresos). Para evitar ambiguedad, es preferible usar un valor explícito en la sentencia de retorno, bien una variable, ó bien una constante, como `return 0`.

El conjunto completo de opciones de aviso incluidas en ‘`-Wall`’ puede encontrarse en el Manual de Referencia de GCC “*Using GCC*” (véase [Lectura adicional], página 131). Las opciones incluidas en ‘`-Wall`’ tienen la característica común de informar de construcciones son siempre erróneas, o pueden ser fácilmente reescritas en un inambiguo camino correcto. Esta es la razón de que éstos sean tan útiles —cualquier aviso producido por ‘`-Wall`’ puede ser tomado como una indicación de un potencial serio problema.

3.5 Opciones de aviso adicionales

GCC proporciona muchas otras opciones de aviso que no son incluidas en '`-Wall`' pero que con frecuencia son útiles. Generalmente estas producen avisos para el código fuente que puede ser técnicamente válido, pero puede causar problemas. El criterio para estas opciones está basado en la experiencia de errores comunes — estos no son incluidos en '`-Wall`' sólo indican posibles problemas o código "sospechoso".

Debido a que estos avisos pueden ser resueltos con código válido no es necesario compilar con estas opciones todo el tiempo. Es más apropiado usarlas periódicamente y revisar los resultados, comprobando cualquier cosa inesperada, o habilitarlas para algunos programas o ficheros.

'`-W`' Esta es una opción general similar a '`-Wall`' que avisa acerca de una selección de errores comunes de programación, tales como funciones que no devuelven un valor, y comparaciones entre valores con y sin signo. Por ejemplo, la siguiente función comprueba si un entero sin signo es negativo (lo cual es imposible, claro):

```
int
foo (unsigned int x)
{
  if (x < 0)
    return 0;  /* no puede ocurrir */
  else
    return 1;
}
```

Tras compilar esta función con '`-Wall`' no se produce ningún aviso,

```
$ gcc -Wall -c w.es.c
```

pero da un aviso con '`-W`':

```
$ gcc -W -c w.c
w.c: In function 'foo':
w.c:4:3: warning: comparison of unsigned
expression < 0 is always false [-Wtype-limits]
```

En la práctica, las opciones '`-W`' y '`-Wall`' se usan juntas.

‘`-Wconversion`’

Esta opción avisa acerca de conversiones implícitas de tipo que podrían causar resultados inesperados, tales como conversiones entre tipos reales y enteros, entre tipos con y sin signo y entre tipos de diferente tamaño (por ej. enteros long y short). Las conversiones pueden ocurrir en expresiones y asignaciones, y en llamadas a funciones si los tipos de los argumentos no concuerdan con aquellos especificados en el prototipo.

Por ejemplo, el valor entero de la función absoluto `int abs(int i)` es fácilmente confundido con la correspondiente función de coma flotante `double fabs(double x)`. Esto puede traer resultados incorrectos, como muestra el siguiente programa:

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

int
main (void)
{
  double x = -3.14;
  double y = abs(x); /* debe ser fabs(x) */
  printf ("x = %g |x| = %g\n", x, y);
  return 0;
}
```

Tras compilar esta función con ‘`-Wall`’ no se produce ningún aviso,

```
$ gcc -Wall wabs.es.c
$ ./a.out
x = -3.14 |x| = 3  (incorrecto)
```

pero da un aviso con ‘`-Wconversion`’:

```
$ gcc -Wall -Wconversion wabs.c
wabs.c: In function 'main':
wabs.c:8:3: warning: conversion to 'int' from
'double' may alter its value [-Wconversion]
```

La opción ‘`-Wconversion`’ también captura errores tales como la asignación de un valor negativo a una variable sin signo, como en el siguiente código,

```
unsigned int x = -1;
```

Esto está permitido técnicamente por el estándar ANSI/ISO C (con el entero negativo siendo convertido a entero positivo, de acuerdo a la representación de la máquina) pero podría ser un simple error de programación. Si se necesita realizar tal conversión se puede usar un cast explícito, tal como `(unsigned int)-1`, para evitar avisos con esta opción. En máquinas con complemento a dos el cast de -1 da el máximo número que puede ser representado por un entero sin signo.

'`-Wshadow`'

Esta opción avisa acerca de la redeclaración de un nombre de variable en un contexto en el que ha sido declarado. Esto define una variable *oculta*, y causa la confusión de qué ocurrencia de la variable corresponde a qué valor.

La siguiente función declara una variable local `y` que oculta la declaración en el cuerpo de la función:

```
double
test (double x)
{
  double y = 1.0;
  {
    double y;
    y = x;
  }
  return y;
}
```

Esto es ANSI/ISO C válido, y el valor devuelto es 1. La ocultación de la variable `y` podría parecer (de manera incorrecta) que el valor devuelto es `x`, al mirar la línea `y = x` (especialmente en una larga y complicada función).

La ocultación también puede ocurrir en nombres de función. Por ejemplo, el siguiente programa intenta definir una variable `sin` que oculta la función estándar `sin(x)`.

```
double
sin_series (double x)
{
  /* series de expansion para x pequenas */
```

```
    double sin = x * (1.0 - x * x / 6.0);
    return sin;
}
```

Este error será detectado con la opción '`-Wshadow`'.

'`-Wcast-qual`'
: Esta opción avisa de punteros que son transformados para eliminar un calificador de tipo tal como `const`. Por ejemplo, la siguiente función descarta el calificador `const` desde su argumento de entrada, permitiéndole ser sobreescrito:

```
void
f (const char * str)
{
  char * s = (char *)str;
  s[0] = '\0';
}
```

La modificación de los contenidos originales de `str` es una violación de la propiedad `const`. Esta opción avisará de la inapropiada transformación de la variable `str` que permite que la cadena sea modificada.

'`-Wwrite-strings`'
: Esta opción implícitamente da a todas las constantes definidas en el programa un calificador `const`, causando un aviso en tiempo de compilación si hay un intento de sobreescribirlo. El resultado de modificar una cadena constante no está definido por el estándar ANSI/ISO, y el uso de constantes de cadena escribibles está obsoleto en GCC.

'`-Wtraditional`'
: Esta opción avisa acerca de partes de código que deberían ser interpretadas de manera diferente por un compilador ANSI/ISO y un compilador "tradicional" pre-ANSI.[6] Cuando se mantiene software legado puede ser necesario investigar si en el código original se pretendía la interpretación ANSI/ISO o la tradicional para entender los avisos generados por esta opción.

[6] La forma tradicional del lenguaje C fué descrita en el manual de referencia original de C "*The C Programming Language (First Edition)*" por Kernighan y Ritchie.

Las opciones anteriores producen mensajes de avisos de diagnóstico, pero permiten la compilación para continuar y producir un fichero objeto o ejecutable. Para programas grandes esto puede ser deseable para capturar todos los avisos deteniendo la compilación siempre que se genera un aviso. La opción '`-Werror`' cambia el comportamiento por defecto al convertir los avisos en errores, parando la compilación siempre y cuando ocurra un aviso.

3.6 Opciones de aviso recomendadas

Las siguiente opciones son una buena elección para encontrar problemas en programas C y C++:

```
$ gcc -ansi -pedantic -Wall -W -Wconversion
    -Wshadow -Wcast-qual -Wwrite-strings
```

Aunque esta lista no es exhaustiva, el uso regular de estas opciones capturará muchos errores comunes.

4 Usando el preprocesador

Este capítulo describe el uso del preprocesador C GNU `cpp`, que forma parte del paquete GCC. Este preprocesador expande las macros en los ficheros fuentes antes de ser compilados. Es invocado de forma automática cada vez que GCC procesa un programa C o C++.[1]

4.1 Definiendo macros

El siguiente programa demuestra el uso habitual del preprocesador C. Se usa la instrucción condicional del preprocesador `#ifdef` para comprobar si la macro ya fué definida:

```
#include <stdio.h>

int
main (void)
{
#ifdef TEST
  printf ("Modo test\n");
#endif
  printf ("Ejecutando...\n");
  return 0;
}
```

Cuando la macro está definida, el preprocesador incluye el código correspondiente antes del comando de cierre `#endif`. En este ejemplo la macro comprobada es llamada `TEST`, y la parte condicional del código fuente es la instrucción `printf` que imprime el mensaje "`Modo test`".

La opción de `gcc` '`-D`*NAME*' define una macro del preprocesador `NAME` desde la línea de comandos. Si el programa anterior es compilado con la opción de línea de comandos '`-DTEST`', la macro `TEST` será definida y el ejecutable resultante imprimirá ambos mensajes:

```
$ gcc -Wall -DTEST dtest.es.c
$ ./a.out
Modo test
Ejecutando...
```

[1] En recientes versiones de GCC el preprocesador está integrado dentro del compilador, aunque un comando por separado `cpp` está siempre incluido.

Si el programa es compilado sin la opción '`-D`' entonces el mensaje "`Modo test`" se omite del código fuente después del preproceso, y el ejecutable final no incluirá el código para él:

```
$ gcc -Wall dtest.es.c
$ ./a.out
Ejecutando...
```

Las macros están generalmente indefinidas, a menos que sea especificada en la línea de comandos, o en un archivo fuente (o un archivo de cabecera de librería) con `#define`. Algunas macros son definidas de forma automática por el compilador —habitualmente usan un espacio de nombres reservado comenzando con un prefijo de doble subrayado '`__`'.

El conjunto completo de macros predefinidas por el preprocesador puede ser listado ejecutanto el preprocesador `cpp` con la opción '`-dM`' en un archivo vacio:

```
$ cpp -dM /dev/null
#define __i586 1
#define __WINT_MAX__ 4294967295U
#define __ORDER_LITTLE_ENDIAN__ 1234
#define __SIZE_MAX__ 4294967295U
.......
```

Obsérvese que este listado incluye un pequeño número de macros específicas del sistema definidas por `gcc` que no usan el prefijo de doble subrayado. Estas macros no estándar pueden ser deshabilitadas con la opción '`-ansi`' de `gcc`.

4.2 Macros con valor

Cuando es definida, una macro puede además tener asignado un valor. Este valor es sustituido dentro del código fuente en cada lugar donde aparezca la macro. El siguiente programa usa la macro `NUM`, para representar que será impreso:

```
#include <stdio.h>

int
main (void)
{
  printf ("El valor de NUM es %d\n", NUM);
  return 0;
}
```

Observe las macros no se expanden dentro de la cadena —solo la ocurrencia de `NUM` fuera de la cadena es sustituida por el preprocesador.

Para definir una macro con un valor, se usa la opción '`-D`' en la línea de comandos en la forma '`-D`*`NAME`*`=`*`VALUE`*'. Por ejemplo, la siguiente línea de comandos define `NUM` a 100 mientras se compila el programa anterior:

```
$ gcc -Wall -DNUM=100 dtestval.es.c
$ ./a.out
El valor de NUM es 100
```

Este ejemplo usa un número, pero una macro puede tomar valores de cualquier forma. Cuando el valor de un macro está asignado, éste es insertado directamente dentro del código fuente donde la macro aparezca. Por ejemplo, la siguiente definición expande las ocurrencias de `NUM` a `2+2` durante el preprocesamiento:

```
$ gcc -Wall -DNUM="2+2" dtestval.es.c
$ ./a.out
El valor de NUM es 4
```

Una vez que el preprocesador ha realizado la sustitución `NUM` $\mapsto$ `2+2` es equivalente a compilar el siguiente programa:

```
#include <stdio.h>

int
main (void)
{
  printf ("El valor de NUM es %d\n", 2+2);
  return 0;
}
```

Observe que una buena idea es encerrar las macros entre paréntesis siempre que formen parte de una expresión. Por ejemplo, el programa siguiente usa paréntesis para asegurar la correcta precedencia para la multiplicación `10*NUM`:

```
#include <stdio.h>

int
main (void)
{
  printf ("Diez veces NUM es %d\n", 10 * (NUM));
  return 0;
}
```

Con estos paréntesis, se produce el resultado esperado cuando se compila con la misma línea de comandos anterior:

```
$ gcc -Wall -DNUM="2+2" dtestval3.es.c
$ ./a.out
Diez veces NUM es 40
```

Sin los paréntesis, el programa hubiera producido el valor `22` desde la forma literal de la expresión `10*2+2 = 22`, en vez del valor deseado `10*(2+2) = 40`.

Cuando una macro es definida solo con la opción '`-D`', `gcc` usa el valor por defecto `1`. Por ejemplo, compilando el programa test original con la opción '`-DNUM`' se genera un ejecutable que produce la siguiente salida:

```
$ gcc -Wall -DNUM dtestval.es.c
$ ./a.out
El valor de NUM es 1
```

Una macro puede ser definida con un valor vacio usando comillas en la línea de comandos, `-D`*`NAME`*`=""`. Tal macro será tratada como *definida* por las instrucciones condicionales, tal como `#ifdef`, pero no se expande en nada.

Una macro que contenga comillas se puede definir usando el carácter de escape de la shell. Por ejemplo, la opción `-DMESSAGE='"¡Hola, Mundo!"'` define una macro `MESSAGE` que se expandirá en la secuencia de caracteres `"¡Hola, Mundo!"`. Los caracteres de escape `'...'` protegen los comillas de la cadena `"¡Hola, Mundo!"`. Para una explicación de los diferentes tipos de comillas y secuencias de escape usandas en la shell vea "*GNU Bash Reference Manual*",[Lectura adicional], página 131.

4.3 Preprocesando archivos fuentes

Es posible ver el efecto del preprocesador en los archivos fuentes directamente, usando la opción '`-E`' de `gcc`. Por ejemplo, el fichero siguiente define y usa la macro `TEST`:

```
#define TEST "!Hola, Mundo!"
const char str[] = TEST;
```

Si este fichero se llama '`test.es.c`' el efecto del preprocesador puede ser visto con la siguiente línea de comandos:

```
$ gcc -E test.es.c
# 1 "test.es.c"
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "test.es.c"

const char str[] = "¡Hola, Mundo!";
```

La opción '`-E`' causa que `gcc` ejecute el preprocesador, muestre la salida expandida, y termine sin compilar el código fuente resultante. El valor de la macro `TEST` es sustituido directamente en la salida, produciendo la secuencia de caracteres `const char str[] = "¡Hola, Mundo!" ;`.

El preprocesador además inserta líneas de registro del archivo fuente y el número de línea en la forma `# `*`número-de-línea`* `"`*`archivo-fuente`*`"`, para ayudar en la depuración y permite al compilador mostrar los mensajes de error referentes a esta información. Estas líneas no afectan al programa en sí.

La posibilidad de ver los archivos fuentes precompilados puede resultar útil para examinar los efectos de los archivos de cabeceras del sistema, y encontrar las declaraciones de las funciones del sistema. El siguiente programa incluye el archivo de cabecera '`stdio.h`' para obtener la declaración de la función `printf`:

```
#include <stdio.h>

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

Es posible examinar las declaraciones incluidas en los archivos de cabeceras preprocesando el archivo con `gcc -E`:

```
$ gcc -E hola.es.c
```

En un sistema GNU, esto produce una salida similar a lo siguiente:

```
# 1 "hola.es.c"
# 1 "/usr/include/stdio.h" 1 3
extern FILE *stdin;
extern FILE *stdout;
extern FILE *stderr;

extern int fprintf (FILE * __stream,
                    const char * __format, ...)  ;
extern int printf (const char * __format, ...)  ;

 [ ... declaraciones adicionales ... ]
# 1 "hola.es.c" 2
int
main (void)
{
```

```
  printf ("¡Hola, Mundo!");
  return 0;
}
```

Los archivos de cabeceras del sistema preprocesados suelen generar una extensa salida. Ésta puede ser redirigida a un archivo, o almacenada más convenientemente usando la opción '`-save-temps`':

```
$ gcc -c -save-temps hola.es.c
```

Después de ejecutar este comando, la salida del preprocesador estará disponible en el archivo '`hola.i`'. La opción '`-save-temps`' además almacena los archivos de ensamblado '`.s`' y los archivos objetos '`.o`' además de los archivos preprocesados '`.i`'.

5 Compilando para depuración

Normalmente, un archivo ejecutable no contiene ninguna referencia del archivo fuente original, tal como nombres de variables o números de línea —el archivo ejecutable es una simple secuencia de instrucciones en código máquina producidas por el compilador. Esto es insuficiente para depuración, ya que no es una tarea fácil la forma de encontrar la causa del fallo en un programa.

GCC proporciona la *opción de depuración* '`-g`' para guardar información de depuración en los archivos objetos y ejecutables. Esta información de depuración permite seguir la pista de los errores desde una instrucción específica a la correspondiente línea en el código fuente original. La ejecución de un programa compilado con '`-g`' puede ser seguido por un depurador, tal como el Depurador GNU `gdb` (para más informacón vea "*Debugging with GDB: The GNU Source-Level Debugger*", [Lectura adicional], página 131). Usando un depurador es posible examinar los valores de las variables mientras está en ejecución un programa.

La opción de depuración del compilador funciona almacenando los nombres de funciones y variables y las líneas del código fuente en una *tabla de símbolos* en el archivo objeto o ejecutable.

5.1 Examinando archivos core

Adicionalmente a permitir que un programa se ejecute en un depurador, un importante beneficio de la opción '`-g`' es que permite examinar la causa del fallo de un programa a través de un "core dump".

Cuando un programa termina de forma anormal (p.e. un fallo) el sistema operativo puede escribir un *core file* (habitualmente llamado '`core`') que contiene el estado de memoria del programa en el momento del fallo. Este fichero es a menudo referido como un *core dump*.[1] Combinada con la información de la tabla de símbolos producida por la opción '`-g`', se puede usar el core dump para encontrar la línea donde el programa se paró, y los valores de las variables en este punto.

Esto es muy usado tanto durante el desarrollo de software como después del desarrollo —permite investigar los problemas cuando un programa falla "a pié de campo".

[1] Esta terminología data de los tiempos de las memorias magnéticas.

Aquí hay un simple ejemplo de un programa que contiene un fallo de acceso inválido a memoria, el cual se usará para producir un archivo core dump:

```
int foo (int *p);

int
main (void)
{
  int *p = 0;   /* puntero nulo */
  return foo (p);
}

int
foo (int *p)
{
  int y = *p;
  return y;
}
```

Este programa intenta referenciar un puntero nulo `p`, lo cual es una operación inválida. En la mayoría de lo sistemas operativos, esto causa un fallo del programa.[2]

Para que sea posible encontrar la causa de un posterior fallo, se necesitará compilar el programa con la opción '`-g`':

```
$ gcc -Wall -g null.es.c
```

Observe que el puntero nulo sólo causará un problema en tiempo de ejecución, así la opción '`-Wall`' no producirá ningún aviso.

Haciendo correr el archivo ejecutable en un sistema x86 GNU/Linux causará que el sistema operativo lo finalice de forma anormal:

```
$ ./a.out
Segmentation fault (core dumped)
```

Siempre que aparezca el mensaje '`core dumped`', el sistema operativo producirá un archivo denominado '`core`' en el directorio en curso.[3] Este archivo core contiene una copia completa de las páginas

[2] Históricamente, un puntero nulo correspondía con la posición 0 de memoria, la cual es normalmente restringida por el kernel del sistema operativo. En la práctica no siempre es así como se trata un puntero nulo, pero el resultado es habitualmente el mismo.

[3] Algunos sistemas, tales como FreeBSD y Solaris, se pueden configurados para escribir los archivos core en directorios específicos, p.e. '`/var/coredumps/`', usando los comandos `sysctl` y `coreadm`.

de memoria que el programa mantenía en el momento de su finalización. A propósito, el término *segmentation fault* se refiere al hecho de que el programa intentó un acceso a un "segmento" de memoria restringido fuera del área de memoria que tiene reservada para él.

Algunos sistemas están configurados para no escribir archivos core por defecto, debido a que los archivos pueden ser muy grandes y rápidamente ocupan el espacio disponible en los discos de un sistema. En la shell *GNU Bash* el comando `ulimit -c` controla el tamaño máximo de los archivos core. Si el límite de tamaño es 0, no se producirán archivos core. El tamaño límite actual se puede mostrar introduciendo el siguiente comando:

```
$ ulimit -c
0
```

Si el resultado es cero, como se mostró anteriormente, entonces puede ser incrementado con el siguiente comando para permitir escribir archivos core de cualquier tamaño:[4]

```
$ ulimit -c unlimited
```

Observe que esta configuración solo aplica en la shell actual. Para poner límite en futuras sesiones el comando se deberá poner en el archivo adecuado de entrada al sistema, tal como '`.bash_profile`' para la Bash shell GNU.

Los archivos core pueden ser cargados por el Depurador GNU `gdb` con el siguiente comando:

```
$ gdb FICHERO-EJECUTABLE FICHERO-CORE
```

Observe que tanto el archivo ejecutable original como el archivo core son requeridos para la depuración —no es posible depurar un archivo core sin el correspondiente ejecutable. En este ejemplo, podemos cargar los archivos ejecutable y core con el comando:

```
$ gdb a.out core
```

El depurador inmediatamente comienza a imprimir información de diagnóstico, y muestra un listado de la línea donde el programa ha fallado (línea 13):

[4] Este ejemplo usa el comando `ulimit` en la Bash shell GNU. En otros sistemas operativos el uso de `ulimit` puede variar, y tener un nombre diferente (la shell `tcsh` a su vez, usa el nombre `limit`). El tamaño límite de los archivos core puede ser puesto a un valor específico en kilobytes.

```
$ gdb a.out core
Core was generated by `./a.out'.
Program terminated with signal 11, Segmentation fault
Reading symbols from /lib/libc.so.6...done.
Loaded symbols for /lib/libc.so.6
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2
#0  0x080483ed in foo (p=0x0) at null.es.c:13
13        int y = *p;
(gdb)
```

La línea final `(dbg)` es el prompt del depurador GNU —indica que se pueden introducir posteriores comandos en este punto.[5]

Para investigar la causa del fallo, mostramos el valor del puntero `p` usando el comando `print` del depurador:

```
(gdb) print p
$1 = (int *) 0x0
```

Se muestra que `p` es un puntero nulo (`0x0`) del tipo '`int *`', así nosotros sabemos que la referencia a él con la expresión `p` en esta línea a causado el fallo.

5.2 Mostrando un rastreo

El depurador puede mostrar las llamadas a funciones y sus argumentos hasta el punto actual de ejecución —esto es llamado *pila de rastreo* (stack backtrace) y se muestra con el comando `backtrace`:

```
(gdb) backtrace
#0  0x080483ed in foo (p=0x0) at null.es.c:13
#1  0x080483d9 in main () at null.es.c:7
```

En este caso, la pila de llamadas muestra que el fallo ocurrió en la línea 13 después de que la función `foo` fuera llamada desde `main` con el argumento `p=0x0` en la línea 7 de '`null.es.c`'. Es posible moverse por los diferentes niveles de la pila de llamadas, usando los comandos del depurador `up` y `down`.

[5] Los detalles de la salida de diagnóstico pueden variar ligeramente dependiendo de que versión de `gdb` esté usando.

5.3 Poniendo un punto de ruptura

Un *punto de ruptura* (breakpoint) detiene la ejecución del programa y devuelve el control al depurador, donde las variables y la memoria pueden ser examinadas antes de continuar. Los puntos de ruptura pueden ponerse en funciones específicas, líneas o localizaciones de memoria con el comando `break`.

Para poner un breakpoint en una función, usa el comando `break` *`function-name`*. Por ejemplo, el siguiente comando pone un breakpoint al inicio de la función `main` en el programa anterior:

```
$ gdb a.out
(gdb) break main
Breakpoint 1 at 0x80483c6: file null.es.c, line 6.
```

El depurador tomará el control del programa cuando la función `main` sea llamada. Como la función `main` es la primera función en ser ejecutada en un programa C, el programa se detendrá inmediatamente cuando se ejecute:

```
(gdb) run
Starting program: a.out
Breakpoint 1, main () at null.es.c:6
6         int *p = 0;   /* null pointer */
(gdb)
```

La consola muestra la línea que será ejecutada en siguiente lugar (el número de línea se muestra a la izquierda). El breakpoint detiene el programa *antes* de ejecutar la línea, en este instante el puntero `p` está indefinido y no ha sido puesto a cero aún.

5.4 Paso a paso a través de un programa

Para avanzar y ejecutar la línea mostrada anteriormente, usa el comando `step`:

```
(gdb) step
7         return foo (p);
```

Después de ejecutar la línea 6, el depurador muestra la siguiente línea que será ejecutada. El puntero `p` está ahora puesto a cero (null):

```
(gdb) print p
$1 = (int *) 0x0
```

El comando `step` continuará la ejecución del programa interactivamente a través de cualquier función que sea invocada en la línea actual. Si desea avanzar sin hacer un seguimiento de estas llamadas a funciones, use en cambio el comando `next`.

5.5 Modificando variables

Para arreglar temporalmente el fallo del puntero nulo descubierto anteriormente, podemos cambiar el valor de `p` en el programa en ejecución usando el comando `set variable`.

Las variables pueden ser inicializadas a un valor específico, o ser el resultado de una expresión, la cual puede incluir llamadas a funciones. Esta poderosa característica permite a las funciones de un programa ser testeadas interactivamente a través del depurador.

En este caso, interactivamente, se reservará algo de memoria para el puntero `p` usando la función `malloc`, almacenando el valor 255 en la localización de memoria resultante:

```
(gdb) set variable p = malloc(sizeof(int))
(gdb) print p
$2 = (int *) 0x40013f98 (dirección reservada por malloc)
(gdb) set variable *p = 255
(gdb) print *p
$3 = 255
```

Si ahora continuamos avanzando en el programa con el nuevo valor de `p` el anterior fallo de segmentación no se producirá:

```
(gdb) step
foo (p=0x40013f98) at null.es.c:13
13        int y = *p;
(gdb) step
14        return y;
```

5.6 Continuando la ejecución

El comando `finish` continua la ejecución hasta el final de la actual función, mostrando el valor de retorno:

```
(gdb) finish
Run till exit from #0 0x08048400 in foo (p=0x40013f98)
at null.es.c:15
0x080483d9 in main () at null.es.c:7
7         return foo (p);
Value returned is $13 = 255
```

Para continuar la ejecución hasta que el programa termine (o hasta el siguiente breakpoint) use el comando `continue`,

```
(gdb) continue
Continuing.
Program exited with code 0377.
```

Observe que el código de salida se muestra en notación octal (0377 base 8 = 255 en base 10).

5.7 Más información

Por simplicidad, los ejemplos de este capítulo demuestran como usar `gdb` en la línea de comandos. Existen poderosas herramientas como Emacs en modo GDB (*M-x gdb*), DDD o INSIGHT, interfaces gráficos de `gdb`. Los enlaces a estos programas se puede encontrar en la página web del editor de este libro.[6]

Una completa descripción de todos los comandos disponibles en `gdb` se puede encontrar en el manual "*Debugging with GDB: The GNU Soure-Level Debugger*" (véase [Lectura adicional], página 131).

[6] http://www.network-theory.co.uk/gcc/intro/

6 Compilando con optimización

GCC es un *compilador de optimización*. Suministra un ámplio rango de opciones que ayudan a incrementar la velocidad, o reducir el tamaño, de los archivos ejecutables generados.

La optimización es un proceso complejo. Para cada comando de alto nivel en el código fuente normalmente hay muchas combinaciones posibles de instrucciones máquina que pueden ser usadas para alcanzar los resultados finales apropiados. El compilador debe considerar todas estas posibilidades y elegir entre ellas.

En general, diferente código es generado para diferentes procesadores, ya que usan ensambladores y lenguajes máquina incompatibles. Cada tipo de procesador tiene sus propias características —Algunas CPUs proporcionan un gran número de *registros* para manejar los resultados intermedios de cálculos, mientras que otros muchos deben almacenar y recuperar los resultados intermedios de la memoria. El código apropiado debe generarse en cada caso.

Además, diferentes instrucciones necesitan diferentes cantidades de tiempo, dependiendo de como fueron ordenadas. GCC, cuando compila con optimización, toma estos factores en cuenta e intenta producir el ejecutable más rápido para un sistema determinado.

6.1 Optimización a nivel de fuentes

La primera forma de optimización usada por GCC sucede a nivel de código fuente, y no requiere ningún conocimiento de las instrucciones máquina. Hay muchas técnicas de optimización a nivel de código fuente —esta sección describe dos formas comunes: *eliminación de subexpresión común* y *expansión de función en línea*.

6.1.1 Eliminación de subexpresión común

Un método de optimización a nivel de código fuente fácil de comprender implica el tratamiento de expresiones en el código fuente con menor número de instrucciones, reutilizando resultados ya calculados. Por ejemplo, la siguiente asignación:

```
x = cos(v)*(1+sin(u/2)) + sin(w)*(1-sin(u/2))
```

puede ser reescrita con una variable temporal `t` y así eliminar evaluaciones extras innecesarias del término `sin(u/2)`:

```
t = sin(u/2)
x = cos(v)*(1+t) + sin(w)*(1-t)
```

Esta reescritura es denominada *eliminación de subexpresiones comunes* (CSE)[1], y se realiza de forma automática cuando la optimización está activa.[2] La eliminación de subexpresiones comunes es beneficiosa porque simultáneamente incrementa la velocidad y reduce el tamaño del código.

6.1.2 Expansión de función en línea

Otro tipo de optimización a nivel de código fuente, denominada *expansión de función en línea*, incrementa la eficiencia de la llamadas a funciones frecuentes.

Cuando una función se invoca, se requiere una cierta cantidad de tiempo extra de CPU para llevar a cabo la llamada: se almacenan los argumentos de la función en los apropiados registros y localizaciones de memoria, se salta al inicio de la función (trayendo las adecuadas páginas de memoria virtual en memoria física ó en la caché de la CPU si fuera necesario), comienza la ejecución del código, y se vuelve al punto original de ejecución donde la llamada a la función se completa. Este trabajo adicional es llamado *sobrecarga de la llamada a función*. La *expansión de función en línea* elimina esta sobrecarga sustituyendo llamadas a funciones por el código mismo de la función (conocido como poner el código *en-línea*).

En la mayoría de los casos, la sobrecarga de las llamadas a funciones es una fracción insignificante del total de tiempo de ejecución de un programa. Puede ser significativo sólo cuando las funciones contienen relativamente pocas instrucciones y emplean una fracción sustancial de tiempo de ejecución —en este caso la sobrecarga llega a ser una mayor proporción del tiempo de ejecución.

La expansión de funciones en línea son siempre favorables si hay un único punto de invocación a la función. Esto es incondicionalmente mejor si la invocación a una función requiere más instrucciones (memoria) que mover el cuerpo de la función en linea. Esto es una situación habitual para una simple función de acceso en C++, lo cual puede beneficiar mucho a las funciones en línea. Por otra parte, las funciones en línea pueden facilitar optimizaciones futuras,

[1] Nota del traductor: Siglas en inglés de 'Common Subexpression Elimination'

[2] Los valores introducidos temporalmente por el compilador durante la eliminación de subexpresiones comunes son usados sólo internamente, y no afectan a las variables reales. El nombre de la variable temporal `t` mostrado anteriormente sólo es a modo ilustrativo

como la eliminación de subexpresiones comunes, mezclando varias funciones separadas en un sencilla función más grande.

La siguiente función `sq(x)` es un típico ejemplo de una función que se beneficiaría de la expansión de función en línea. Calcula x^2, el cuadrado del argumento x:

```
double
sq (double x)
{
  return x * x;
}
```

Esta función es pequeña, así la sobrecarga de la llamada es comparable al tiempo tomado en ejecutar una simple multiplicación fuera de la función misma. Si esta función se invoca dentro de un bucle, tal y como sucede más abajo, entonces la sobrecarga de la llamada a la función llegaría a ser sustancial:

```
for (i = 0; i < 1000000; i++)
  {
     sum += sq (i + 0.5);
  }
```

La optimización con expansión de función en línea reemplaza el interior del bucle del programa con el cuerpo de la función, dando el siguiente código:

```
for (i = 0; i < 1000000; i++)
  {
    double t = (i + 0.5);  /* variable temporal */
    sum += t * t;
  }
```

La eliminación de la llamada a la función y realizar la multiplicación *en línea* permite al bucle ejecutarse con máxima eficiencia.

GCC selecciona funciones para poner expandir en línea usando un número de heurísticas, tal como funciones que son adecuadamente pequeñas. Como optimización, las funciones en línea son llevadas a cabo sólo dentro del archivo objeto. La palabra reservada `inline` pueder ser usada para requerir explícitamente que una función específica deba ser tratada en línea como sea posible, incluyendo sus usos en otros archivos.[3] El Manual de Referencia de GCC "*Using GCC*" suministra un completo detalle de la palabra reservada `inline`, y emplea los calificadores `static` y `extern` para controlar el enlazado de las funciones en línea explícitamente (véase [Lectura adicional], página 131).

[3] En este caso, la definición de la función en línea debe ser hecha en otro archivo (p.e. en un archivo de cabeceras).

6.2 Dilema velocidad-espacio

Mientras que algunas formas de optimización, tales como la eliminación de subexpresiones comunes, son capaces simultáneamente de incrementar la velocidad y reducir el tamaño de un programa, otros tipos de optimización producen código más rápido a costa de incrementar el tamaño del ejecutable. Esta elección entre velocidad y espacio se denomina *dilema velocidad-espacio*. Las optimizaciones con el *dilema velocidad-espacio* se pueden usar en sentido inverso para hacer ejecutables pequeños, a cambio de hacerlos correr más lentos.

6.2.1 Desenrollado de bucle

Un primer ejemplo de optimización con el *dilema velocidad-espacio* es *Desenrollado de bucle*. Esta forma de optimización incrementa la velocidad de los bucles eliminando la condición de terminación del bucle en cada iteración. Por ejemplo, el siguiente bucle de 0 a 7 comprueba la condición `i < 8` en cada iteración:

```
for (i = 0; i < 8; i++)
  {
    y[i] = i;
  }
```

Al final del bucle, esta condición ha sido comprobada 9 veces, y gran parte del tiempo de ejecución fué gastado en la comprobación.

Una forma más eficiente de escribir el mismo código es simplemente *desenrollar el bucle* y ejecuta las instrucciones directamente:

```
y[0] = 0;
y[1] = 1;
y[2] = 2;
y[3] = 3;
y[4] = 4;
y[5] = 5;
y[6] = 6;
y[7] = 7;
```

Este tipo de código no requiere ningún test, y se ejecuta a la máxima velocidad. Una vez que cada asignación es independiente, permite ser compilado para usar procesamiento paralelo cuando es soportado. Desenrollado de bucle es una optimización que incrementa la velocidad del ejecutable resultante pero generalmente incrementa su tamaño (a menos que el bucle sea muy corto, con sólo una o dos iteraciones, por ejemplo).

Desenrollado de bucle se hace posible cuando el límite superior de un bucle es desconocido, siempre y cuando las condiciones de

inicio y terminación sean manejadas correctamente. Por ejemplo, el mismo bucle con un límite superior arbitrario,

```
for (i = 0; i < n; i++)
  {
    y[i] = i;
  }
```

puede ser reescrito por el compilador como sigue:

```
for (i = 0; i < (n % 2); i++)
  {
    y[i] = i;
  }

for ( ; i + 1 < n; i += 2) /* sin inicializador */
  {
    y[i] = i;
    y[i+1] = i+1;
  }
```

El primer bucle trata el caso `i = 0` cuando `n` es impar, y el segundo bucle trata el resto de iteraciones. Observe que el segundo bucle no usa un inicializador en el primer argumento de la instrucción `for`, justo continúa donde el primer bucle terminó. Las instrucciones del segundo bucle pueden ser paralelizadas, y el total del número de tests se reduce en un factor de 2 (aproximadamente). Factores mayores se pueden alcanzar desenrollando más instrucciones dentro del bucle, a cambio de agrandar el tamaño del código.

6.3 Planificación

El menor nivel de optimización es la *planificación*, en el cual el compilador determina el mejor orden para las instrucciones individuales. La mayoría de las CPUs permiten que una o más nuevas instrucciones comiencen a ejecutarse antes de que otras hallan terminado. Muchas CPUs tienen soporte para *segmentación*, donde múltiples instrucciones se ejecutan en paralelo en la misma CPU.

Cuando la planificación está habilitada, las instrucciones pueden disponer de sus resultados llegando a tener disponible la siguiente instrucción en el tiempo correcto, y a permitir un máximo de ejecución en paralelo. La *planificación* aumenta la velocidad de un ejecutable sin incrementar su tamaño, pero requiere memoria y tiempo adicional en sus procesos de compilación (dada su complejidad).

6.4 Niveles de optimización

Para controlar los tiempos de compilación y el uso de memoria, y la compensación entre velocidad y espacio para el ejecutable resultante, GCC suministra un rango de niveles generales de optimización, numerados desde 0 al 3, así como opciones individuales para tipos específicos de optimización.

El nivel de optimización se elige con la opción '`-O`*`NIVEL`*', donde *`NIVEL`* es un número del 0 al 3. Los efectos de los diferentes niveles de optimización se describen a continuación:

'`-O0`' o sin opción '`-O`' (por defecto)
: Este nivel de optimización no realiza ninguna optimización y compila el código fuente de la forma más sencilla posible. Cada comando en el código fuente es convertido directamente a sus correspondientes instrucciones en el archivo ejecutable, sin reorganización. Esta es la mejor opción a usar cuando se depura un programa y es la opción por defecto si no se especifica una opción de nivel de optimización.

'`-O1`' o '`-O`'
: Este nivel activado es la optimización más frecuente ya que no requiere compensar entre velocidad y espacio. Con esta opción los ejecutables resultantes deberían ser más pequeños y rápidos que con la opción '`-O0`'. La optimización más costosa, como la planificación de instrucciones, no es usada a este nivel.

  Compilar con la opción '`-O1`' puede a menudo tomar menos tiempo que la compilación con la opción '`-O0`', debido a la reducida cantidad de datos que necesitan ser procesados después de una simple optimización.

'`-O2`'
: Esta opción activa más optimizaciones, además de las empleadas por '`-O1`'. Estas optimizaciones adicionales incluyen la planificación de instrucciones. Sólo se emplearán optimizaciones que no requieren compensanción entre velocidad y espacio, así el ejecutable no aumentará de tamaño. Al compilador le costará más compilar programas y necesitará más memoria que con la opción '`-O1`'. Esta opción es la más elegida en el desarrollo de programas, porque suministra un máximo de optimización sin incrementar el tamaño del ejecutable. Es la opción del nivel de optimización por defecto en las versiones de paquetes GNU.

‘`-O3`’ Esta opción activa las más costosas optimizaciones, tales como funciones en-línea, además de todas las optimizaciones de niveles inferiores ‘`-O2`’ y ‘`-O1`’. El nivel de optimización ‘`-O3`’ incrementa la velocidad del ejecutable. Bajo algunas circunstancias donde esta optimización no es posible, esta opción podría hacer un programa muy lento.

‘`-funroll-loops`’
Esta opción activa el desenrollado de bucles, y es independiente de otras opciones de optimización. Se incrementará el tamaño del ejecutable. Si esta opción produce o no un resultado beneficioso será comprobado sobre una base caso por caso.

‘`-Os`’ Esta opción selecciona las optimizaciones que reducen el tamaño del ejecutable. El propósito de esta opción es producir el ejecutable menor posible, para sistemas con restricciones de memoria o espacio de disco. En algunos casos un pequeño ejecutable es más rápido, debido al mejor uso de memoria caché.

Es importante recordar que el beneficio de altos niveles de optimización debe sopesarse contra el coste. El coste de la optimización incluye una gran complejidad en depuración, e incrementa los requerimientos de tiempo y memoria durante la compilación. Para la mayoría de los propósitos es conveniente usar la opción ‘`-O0`’ para depuración, y la opción ‘`-O2`’ para el desarrollo y uso posterior.

6.5 Ejemplos

El siguiente programa será usado para demostrar los efectos de los diferentes niveles de optimización:

```
#include <stdio.h>

double
powern (double d, unsigned n)
{
  double x = 1.0;
  unsigned j;

  for (j = 1; j <= n; j++)
    x *= d;

  return x;
```

```
}

int
main (void)
{
  double sum = 0.0;
  unsigned i;

  for (i = 1; i <= 100000000; i++)
    {
      sum += powern (i, i % 5);
    }

  printf ("sum = %g\n", sum);
  return 0;
}
```

El programa principal contiene un bucle invocando a la función `powern`. Esta función calcula la enésima potencia de un número en coma flotante con repetición de multiplicaciones —esto ha sido elegido así porque es conveniente para funciones en línea y desenrollado de bucles. El tiempo de ejecución de un programa puede ser medido usando el comando `time` en la GNU Bash shell.

Aquí se muestran los resultados para el programa anterior, compilado en un 566 MHz Intel Celeron con 16 KB L1-cache y 128 KB L2-cache, usando GCC 3.3.1 en un sistema GNU/Linux:

```
$ gcc -Wall -O0 optim.es.c -lm
$ time ./a.out
real    0m13.388s
user    0m13.370s
sys     0m0.010s

$ gcc -Wall -O1 optim.es.c -lm
$ time ./a.out
real    0m10.030s
user    0m10.030s
sys     0m0.000s

$ gcc -Wall -O2 optim.es.c -lm
$ time ./a.out
real    0m8.388s
user    0m8.380s
sys     0m0.000s
```

```
$ gcc -Wall -O3 optim.es.c -lm
$ time ./a.out
real    0m6.742s
user    0m6.730s
sys     0m0.000s

$ gcc -Wall -O3 -funroll-loops optim.es.c -lm
$ time ./a.out
real    0m5.412s
user    0m5.390s
sys     0m0.000s
```

La entrada relevante en la salida para comparar la velocidad de los ejecutables resultantes es el tiempo '`user`', el cual da el tiempo de CPU empleado en la ejecución del proceso actual. En otras filas, '`real`' y '`sys`', registran el tiempo total real de los procesos que se ejecutan (incluyendo los tiempos de otros procesos que están usando la CPU) y el tiempo empleado en esperar las llamadas al sistema operativo. Aunque sólo se ha mostrado una ejecución, las pruebas de rendimiento serán ejecutadas varias veces para confirmar los resultados.

En los resultados puede apreciarse en este caso que incrementando el nivel de optimización con '`-O1`', '`-O2`' y '`-O3`' se produce un incremento de la velocidad. Añadiendo la opción '`-funroll-loops`' se produce la mayor rapidez. La velocidad del programa aumenta al doble de la inicial, cuando va desde el código no optimizado al más alto nivel de optimización.

Observe que un pequeño programa como este puede tener considerables variaciones para versiones de sistemas y compiladores. Por ejemplo, en un sistema Mobile 2.0 GHz Intel Pentium 4M la tendencia del resultado usando la misma versión de GCC es similar excepto que el rendimiento con '`-O2`' es ligeramente peor que con '`-O1`'. Esto ilustra un importante punto: las optimizaciones no necesariamente hacen programas más rápidos en todos los casos.

6.6 Optimización y depuración

Con GCC es posible usar optimización en combinación con la opción '`-g`' de depuración. Otros muchos compiladores no permiten esto.

Cuando se usa depuración y optimización juntas, la reorganización interna llevada a cabo por el optimizador puede hacer difícil ver que está haciendo un programa cuando se examinan programas optimizados con el depurador. Por ejemplo, las variables temporales son a menudo eliminadas, y el orden de las instrucciones puede estar cambiado.

Sin embargo, cuando un programa falla inesperadamente, cualquier información de depuración es mejor que nada —así el uso de '`-g`' se recomienda para programas optimizados, tanto para desarrollo como para mantenimiento. La opción '`-g`' de depuración está habilitada por defecto en las versiones de los paquetes GNU, junto con la opción '`-O2`' de optimización.

6.7 Optimización y avisos del compilador

Cuando la optimización está activa, GCC puede producir avisos adicionales que no aparecen cuando se compila sin optimización.

Como parte del proceso de optimización, el compilador examina el uso de todas las variables y sus valores iniciales —esto es llamado un *análisis de flujo de datos*. Ésto forma la base de otras estrategias de optimización, tales como la planificación de instrucciones. El efecto del análisis del flujo de datos es que el compilador puede detectar el uso de variables indefinidas.

La opción '`-Wuninitialized`' (la cual está incluida en '`Wall`') avisa sobre variables que son leidas sin haber sido inicializadas. Sólo funciona cuando el programa es compilado con optimización, de modo que el análisis del flujo de datos está habilitado. La siguiente función contiene un ejemplo con tales variables:

```
int
sign (int x)
{
  int s;

  if (x > 0)
    s = 1;
  else if (x < 0)
    s = -1;

  return s;
```

```
}
```

Esta función funciona correctamente para la mayoría de argumentos, pero tiene un fallo cuando `x` es cero —en este caso el valor de retorno de la variable `s` será indefinido.

Compilando el programa con la opción '`-Wall`' solamente no produce ningún aviso, porque el análisis del flujo de datos no es llevado a cabo sin optimización:

```
$ gcc -Wall -c uninit.es.c
```

Para producir avisos, el programa debe ser compilado con '`-Wall`' y optimización simultáneamente. En la práctica, el nivel de optimización '`-O2`' es necesario para obtener buenos avisos:

```
$ gcc -Wall -O2 -c uninit.c
uninit.c: In function 'sign':
uninit.c:11:3: warning: 's' may be used uninitialized
  in this function [-Wuninitialized]
```

Esto detecta correctamente la posibilidad de que la variable `s` sea empleada sin haber sido definida.

Observe que mientras GCC habitualmente encontrará la mayoría de las variables no inicializadas, se estarán usando heurísticas que ocasionalmente fallarán en algunos casos complicados y otras veces falseando avisos. En esta última situación, es a menudo posible reescribir las líneas relevantes de una forma sencilla que elimine los avisos y mejore la legibilidad del código fuente.

7 Compilando un programa C++

Este capítulo describe el uso de GCC para compilar programas escritos en C++. y las opciones específicas de línea de comandos para este lenguaje.

El compilador GNU C++ suministrado por GCC es un verdadero compilador de C++ —compila código fuente directamente en lenguaje ensamblador. Otros "compiladores" C++ hacen la traducción convirtiendo los programas C++ en programas C, y compilan el programa C resultante usando el compilador existente. Un verdadero compilador C++, tal como GCC, es capaz de proveer mejor soporte para informar los errores, depuración y optimización.

7.1 Compilando un pequeño programa C++

El procedimiento para compilar un programa C++ es el mismo que para un programa C, pero usa el comando `g++` en vez de `gcc`. Ambos compiladores son parte de la colección de compiladores GNU.

Para mostrar el uso de `g++`, aquí está la versión del programa *Hola Mundo* escrita en C++:

```
#include <iostream>

int
main ()
{
  std::cout << "!Hola, mundo!\n";
  return 0;
}
```

El programa puede ser compilado con la siguiente línea de comando:

```
$ g++ -Wall hola.es.cc -o hola
```

El frontend C++ de GCC usa muchas opciones similares al compilador de C `gcc`. Además, soporta opciones adicionales para controlar las características del lenguaje C++, las cuales serán descritas en este capítulo. Observe que el código fuente C++ debe darse en una de las extensiones válidas para archivos C++ '`.cc`', '`.cpp`', '`cxx`' o '`.C`' en vez de la extensión '`.c`' usada por programas C.

El ejecutable resultante puede ser ejecutado de la misma forma que la versión C, simplemente tecleando su nombre:

```
$ ./hola
¡Hola, mundo!
```

El ejecutable produce la misma salida que la versión C del programa usando `std::cout` en vez de la función C `printf`. Todas las opciones usadas en comandos `gcc` en capítulos anteriores pueden aplicarse a `g++` sin cambios, al igual que los procedimientos para compilar y enlazar archivos y librerías (usando `g++` en vez de `gcc`, por supuesto). Una natural diferencia es que la opción '`-ansi`' requiere conformidad con el estándar C++, en vez de con el estándar C, cuando es usado con `g++`.

Observe que los programas que usan archivos objeto C++ deben ser enlazados con `g++`, con objeto de suministrar las librerías necesarias. Intentar enlazar un archivo objeto C++ con el compilador de C `gcc` causará errores del tipo "undefined reference" para las funciones de la librería estándar de C++:

```
$ g++ -Wall -c hola.es.cc
$ gcc hola.o        (debería usar g++)
hola.o: In function ‘main’:
hola.o(.text+0x1b): undefined reference to ‘std::cout’
.....
hola.o(.eh_frame+0x11):
  undefined reference to ‘__gxx_personality_v0’
```

Las referencias indefinidas a funciones internas de librerías en tiempo de ejecución, tales como `__gxx_personality_v0`, son pruebas del enlazado de archivos objetos C++ con `gcc` en vez de con `g++`.[1] Enlazando el mismo archivo objeto con `g++` suministra todas las librerías C++ necesarias y produce un ejecutable que funciona:

```
$ g++ hola.es.o
$ ./a.out
¡Hola, mundo!
```

Un aspecto que algunas veces causa confusión es que `gcc` actualmente compila código fuente C++ cuando detecta la extensión de archivos C++, pero no puede enlazar el archivo objeto resultante.

```
$ gcc -Wall -c hola.es.cc    (correcto, en vez de C++)
$ gcc hola.o
hola.o: In function ‘main’:
hola.o(.text+0x1b): undefined reference to ‘std::cout’
```

Para eliminar este problema, use `g++` consistentemente para programas C++ y `gcc` para programas C.

[1] Puesto que estas funciones son internas, el error mostrado en este ejemplo será diferente en otras versiones del compilador.

7.2 Opciones de compilación en C++

La mayoría de las opciones pueden ser usadas tanto para programas C como C++, pero además hay unas pocas opciones que son específicas de cada lenguaje. Esta sección describe algunas de las opciones adicionales, y las mejoras de las opciones existentes, que están disponibles en `g++`.

'`-Wall`' y '`-W`'
: Cuando se compila con `g++`, las opciones '`-Wall`' y '`-W`' incluyen avisos extras específicos de C++ (los avisos relacionan las funciones miembro con las clases virtuales). El uso de estas opciones es siempre recomendado mientras se depura un programa.

'`-fno-default-inline`'
: Esta opción deshabilita el inlining por defecto de las funciones miembro definidas en los cuerpos de las clases C++. GCC normalmente inlines todas las funciones posibles cuando la optimización está activada, incluso si no se usó explícitamente la palabra reservada `inline`. Elija esta opción si desea controlar el inlining por sí mismo, o si desea poner un punto de ruptura en una función miembro que en otro caso sería inlined (ya que no es posible poner un punto de ruptura en una función inlined).

'`-Weffc++`'
: Esta opción avisa sobre el código C++ que rompe alguna de las directrices dadas en los libros "*Effective C++*" y "*More Effective C++* de Scott Meyers. Por ejemplo, se mostrará un aviso si una clase usa memoria dinámicamente ubicada si no es definida como copia de un constructor y un operador de asignación. Observe que los archivos cabeceras de las librerías estándar no siguen estas directrices, así se puede usar esta opción para un test ocasional de posibles problemas en su propio código en vez de compilar con ella todo el tiempo.

'`-Wold-style-cast`'
: Esta opción destaca cualquier uso de moldes con el estilo de C en programas C++. El lenguaje C++ facilita las palabras reservadas `static_cast`, `dynamic_cast`, `reinterpret_cast` y `const_cast` para manejar moldes y éstos con preferidos a menudo (aunque los moldes al estilo C son permitidos).

7.3 Usando la librería estándar de C++

Una implementación de la librería estándar de C++ es suministrada como parte de GCC. El siguiente programa usa la clase `string` de la libraría estándar para reimplementar el programa *Hola Mundo*:

```
#include <string>
#include <iostream>

using namespace std;

int
main ()
{
  string s1 = "!Hola,";
  string s2 = "Mundo!";
  cout << s1 + " " + s2 << '\n';
  return 0;
}
```

El programa puede ser compilado y ejecutado usando los comandos siguientes:

```
$ g++ -Wall holastr.es.cc
$./a.out
¡Hola, Mundo!
```

Observe que de acuerdo con el estándar C++, los archivos cabeceras de las librerías C++ mismas no usan una extensión de archivo. Las clases en la librería son además definidas dentro del espacio de nombres `std`, así la directiva `using namespace std` es necesaria para acceder a ellas, a menos que el prefijo `std::` sea usado en todas partes (como en la sección anterior).

7.4 Plantillas

Las plantillas facilitan la posibilidad de definir clases C++ que soporten técnicas *genéricas de programación*. Las plantillas se pueden considerar como una clase poderosa de macro. Cuando una plantilla de clase o función es usada con una clase o tipo específico, tal como `float` o `int`, el código correspondiente a la plantilla es compilado con el tipo sustituido en los lugares apropiados.

7.4.1 Usando plantillas de librerías estándar de C++

La librería estándar '`libstdc++`' facilitada con GCC suministra un amplio rango de contenedores de clases genéricas, tales como listas y colas, en adición a algoritmos generales tales como ordenación. Estas clases fueron originalmente parte de la Standard Template Library (STL), la cual era un paquete separado, pero no está incluida en la misma librería estándar C++.

El siguiente programa demuestra el uso de una libraría de plantillas creando una lista de cadenas con la plantilla `list<string>`:

```
#include <list>
#include <string>
#include <iostream>

using namespace std;

int
main ()
{
  list<string> list;
  list.push_back("Hello");
  list.push_back("World");
  cout << "List size = " << list.size() << '\n';
  return 0;
}
```

No son necesarias opciones especiales para usar las plantillas de clases de las librerías estándar; las opciones de la línea de comandos para compilar este programa son las mismas de antes:

```
$ g++ -Wall string.es.cc
$ ./a.out
# items = 2
```

Observe que el ejecutable creado por `g++` usando la librería estándar de C++ enlazará a la librería compartida '`libstdc++`', la cual es suministrada como parte de la instalación por defecto de GCC. Hay varias versiones de esta librería —si distribuye ejecutables usando la librería estándar de C++ necesita asegurar que los destinatarios tienen una versión compatible de '`libstdc++`', o enlazar su programa estáticamente usando la opción de línea de comandos '`-static`'.

7.4.2 Proporcionando sus propias plantillas

Además de las plantillas de clases suministradas por la librería estándar de C++ puede definir sus propias librerías. La forma recomendada de usar plantillas con `g++` es siguiendo el *modelo de compilación de inclusión*, donde las definiciones de plantillas están puestas en los archivos de cabeceras. Este es el método usado por la librería estándar de C++ suministrada con el mismo GCC. Los archivos de cabeceras pueden ser incluidos con '`#include`' en cada archivo fuente donde se necesite.

Por ejemplo, el siguiente archivo de plantilla crea una simple clase `Buffer<T>` que representará un buffer circular tratando objetos del tipo `T`.

```
#ifndef BUFFER_H
#define BUFFER_H

template <class T>
class Buffer
{
public:
  Buffer (unsigned int n);
  void insert (const T & x);
  T get (unsigned int k) const;
private:
  unsigned int i;
  unsigned int size;
  T *pT;
};

template <class T>
Buffer<T>::Buffer (unsigned int n)
{
  i = 0;
  size = n;
  pT = new T[n];
};

template <class T>
void
Buffer<T>::insert (const T & x)
{
  i = (i + 1) % size;
  pT[i] = x;
};
```

```
template <class T>
T
Buffer<T>::get (unsigned int k) const
{
  return pT[(i + (size - k)) % size];
};

#endif /* BUFFER_H */
```

El archivo contiene tanto la declaración de la clase como la definición de las funciones miembro. Esta clase sólo se muestra con propósitos de demostración y no debe ser considerada un ejemplo de buena programación. Obsérvese el uso de *guardas include*, que comprueban la existencia de una macro `BUFFER_H`, asegurando que la definición en el archivo de cabecera será sólo considerada una vez si el archivo es incluido múltiples veces en el mismo contexto.

El programa siguiente emplea la plantilla de clase `Buffer` para crear un buffer de tamaño 10, almacenando los valores en coma flotante 0.25 y 1.25 en el buffer:

```
#include <iostream>
#include "buffer.h"

using namespace std;

int
main ()
{
  Buffer<float> f(10);
  f.insert (0.25);
  f.insert (1.0 + f.get(0));
  cout << "stored value = " << f.get(0) << '\n';
  return 0;
}
```

Las definiciones de la clase de plantilla y sus funciones están incluidas en el archivo fuente del programa con '`#include "buffer.es.h"`' antes de ser usada. El programa puede entonces ser compilado usando la siguiente línea de comandos:

```
$ g++ -Wall tprog.es.cc
$ ./a.out
valor almacenado = 1.25
```

En el momento en que las funciones de la plantilla son usadas en el código fuente, `g++` compila las definiciones apropiadas de los

archivos de cabeceras y pone las funciones compiladas en el correspondiente archivo objeto.

Si una plantilla de función es usada varias veces en un mismo programa será almacenada en más de un archivo objeto. El enlazador GNU asegura que sólo una copia será puesta en el ejecutable final. Otros enlazadores pueden devolver errores tipo "*definición múltiple de símbolo*" cuando aparecen más de una copia de una plantilla de función —un método de trabajo con estos enlazadores es descrito más adelante.

7.4.3 Instanciación explícita de plantillas

Para tomar un completo control sobre la compilación de plantillas con `g++` es posible que se requiera una instanciación explícita de cada ocurrencia de la plantilla, usando la opción '`-fno-implicit-templates`'. Este método no se necesita usando el enlazador GNU —es un alternativa suministrada para sistemas con enlazadores que no pueden eliminar las definiciones duplicadas de funciones de plantillas en los archivos objetos.

Con este enfoque, las funciones de plantillas no son compiladas hasta el momento donde serán usadas, como resultado de la opción '`-fno-implicit-templates`'. En cambio, el compilador busca una instanciación explícita de la plantilla usando la palabra reservada `template` con el tipo específico para forzar su compilación (esto es una extensión de GCC del comportamiento estándar). Estas instanciaciones son normalmente puestas en archivos fuente separados, los cuales serán compilados para hacer los archivos objetos que contengan todas las funciones de plantillas requeridas por el programa. Esto garantiza que cada plantilla aparezca en un único archivo objeto, y que sea compatible con enlazadores que no puedan eliminar definiciones duplicadas en archivos objeto.

Por ejemplo, el siguiente archivo '`templates.es.cc`' contiene una instanciación explícita de la clase `Buffer<float>` usada por el programa `tprog.es.cc` mostrado antes:

```
#include "buffer.h"
template class Buffer<float>;
```

El programa completo puede compilarse y enlazarse usando instanciación explícita con el siguiente comando:

```
$ g++ -Wall -fno-implicit-templates -c tprog.es.cc
$ g++ -Wall -fno-implicit-templates -c templates.es.cc
$ g++ tprog.o templates.o
$ ./a.out
valor almacenado = 1.25
```

El código objeto de todas las funciones de plantillas está contenido en el archivo `templates.o`. No hay código objeto para la función de plantilla en '`tprog.o`' cuando este es compilado con la opción '`-fno-implicit-templates`'.

Si el programa es modificado para usar tipos adicionales, entonces podrían añadirse posteriores instanciaciones explícitas en el archivo '`templates.es.cc`'. Por ejemplo, el siguiente código añade instanciaciones de objetos Buffer conteniendo valores `float`, `double` e `int`:

```
#include "buffer.h"
template class Buffer<float>;
template class Buffer<double>;
template class Buffer<int>;
```

La desventaja de la instanciación explícita es que es necesario conocer qué tipos de plantillas son necesarias en un programa. Para un programa complicado puede ser difícil saberlo por adelantado. Cualquier instanciación de plantilla perdida será detectada en tiempo de enlazado, sin embargo, y añadida a la lista de instanciaciones explícitas, anotando qué funciones están indefinidas.

La instanciación explícita puede ser empleada para hacer librerías de funciones de plantillas precompiladas, creando un archivo objeto que contenga todas las instanciaciones de una función de plantilla (como en el archivo '`templates.es.cc`' anterior). Por ejemplo, el archivo objeto creado desde la instanciación de plantilla anterior contiene el código máquina necesario para las clases Buffer con tipos '`float`', '`double`' e '`int`', y puede ser distribuida como una librería.

7.4.4 La palabra reservada `export`

En el momento de escribir este libro, GCC no soportaba la nueva palabra reservada C++ `export`.

Esta palabra reservada era propuesta como una forma de separar la interfaz con las plantillas de su implementación. Sin embargo añadió complejidad al proceso de enlazado, lo cual en la práctica podría quitar méritos a cualquier ventaja.

La palabra reservada `export` no es ampliamente usada, y la mayoría de los compiladores ni la soportan. El modelo de compilaciín de inclusión descrito anteriormente es recomendado como el más simple y la forma más portable de usar plantillas.

8 Opciones específicas de plataforma

GCC proporciona un rango de opciones específicas de la plataforma para los diferentes tipos de CPUs. Estas opciones controlan las características tales como los modos en coma flotante del hardware, y el uso de instrucciones especiales para diferentes CPUs. Estas pueden seleccionarse con la opción '`-m`' en la línea de comandos, y funciona con todos los frontends de lenguajes GCC, tales como `gcc` y `g++`.

La siguiente sección describe algunas de las opciones disponibles para las plataformas más comunes. Un listado completo de todas las opciones específicas de la plataforma se puede encontrar en el Manual de Referencia GCC, "*Using GCC*" (véase [Lectura adicional], página 131). Soporte a nuevos procesadores es añadido a GCC tan pronto como éste esté disponible, por eso algunas de las opciones descritas en este capítulo no se encuentran en anteriores versiones de GCC.

8.1 Opciones para Intel y AMD x86

Las características más usadas de las familias de procesadores Intel y AMD x86 (386, 486, Pentium, etc) pueden ser controladas con opciones específicas de la plataforma.

En estas plataformas, GCC produce código ejecutable que es compatible por defecto con todos los procesadores de la familia x86 —llegando a retroceder hasta el 386. Sin embargo, es posible compilar para un procesador específico para obtener mejores rendimientos.

Por ejemplo, las versiones recientes de GCC tienen soporte específico para los más nuevos procesadores tales como Pentium 4 y Athlon AMD. Estos pueden ser seleccionados para Pentium 4 con la siguiente opción:

```
$ gcc -Wall -march=pentium4 hola.es.c
```

y para el Athlon:

```
$ gcc -Wall -march=athlon hola.es.c
```

Un listado completo de los tipos de CPUs soportados se puede encontrar en el Manual de Referencia GCC.

El código producido con una opción '`-march=`*`CPU`*' específica puede ser más rápido pero no se ejecutará en otros procesadores de la familia x86. Si planea distribuir archivos ejecutables para uso general en procesadores Intel o AMD, éstos deberían ser compilados sin ninguna opción '`-march`'.

Como alternativa, la opción '`-mcpu=`*`CPU`*' proporciona un compromiso entre velocidad y portabilidad —se genera código que será afinado para un procesador específico, en términos de planificación de instrucciones, pero no emplea ninguna instrucción que no esté disponible en otras CPUs de la faimlia x86.[1] El código resultante será compatible con todas las CPUs, y tiene una velocidad ventajosa en la CPU especificada por '`-mcpu`'. Los ejecutables generados con '`-mcpu`' no pueden lograr el mismo rendimiento que con '`-march`', pero en la práctica puede ser más conveniente.

8.1.1 Extensiones x86

GCC puede obtener ventajas de las instrucciones adicionales en las extenciones MMX, SSE, SSE2, SSE3 y 3dnow de los recientes procesadores Intel y AMD. Las opciones '`-mmmx`', '`-msse`', '`-msse2`', '`-mss3`' y '`-m3dnow`' permiten usar instrucciones extra, permitiendo que múltiples palabras de datos sean procesadas en paralelo. Los ejecutables resultantes sólo funcionarán en procesadores que soporten la apropiada extensión —en otros sistemas pueden fallar con un error `Illegal instruction` o similar.[2]

La opción '`-mfpmath=sse`' instruye a GCC para usar las extensiones SSE para aritmética en coma flotante cuando sea posible. Para que esta opción tenga efecto, las extensiones SSE o SSE2 deberían primero estar habilitadas con '`-msse`' o '`-msse2`'.

Observe que el conjunto de extensiones SSE sólo soporta operaciones de simple precisión —la aritmética de doble precisión es parte de SSE2. Desde que la mayoría de programas C y C++ declaran las variables en coma flotante como `double` en vez de como `float`, las opciones combinadas `-msse2 -mfpmmath=sse` son frecuentemente necesarias. En procesadores de 64 bits estas opciones están disponibles por defecto.

---

[1] En versiones recientes de GCC esta opción ha sido renombrada a '`-mtune`'. La antigua forma '`-mcpu`' continuará funcionando.

[2] En sistemas GNU/Linux, el comando `cat /proc/cpuinfo` mostrará información de la CPU.

8.1.2 Procesadores x86 de 64 bits

AMD ha mejorado el conjunto de instrucciones de 32 bits a un conjunto de instruciones de 64 bits denominado x86-64, el cual es implementado en sus procesadores AMD64.[3] En sistemas AMD64 GCC genera código de 64 bits por defecto. La opción '`-m32`' permite en cambio que sea generado código de 32 bits.

El procesador dispone de varios modelos de memoria diferentes para ejecutar programas en modo de 64 bits. El modelo por defecto es el modelo de código pequeño, el cual permite código y datos de hasta 2 GB de tamaño. En el modelo de código medio permite tamaño ilimitado de datos y puede ser seleccionado con '`-mcmode=medium`'. También hay un modelo de código grande, el cual soporta tamaño ilimitado de código además del tamaño ilimitado de datos. Este no está actualmente implementado en GCC una vez que el modelo de código medio es suficiente para todos los propósitos prácticos —los ejecutables con tamaños superiores a 2 GB en la práctica no se encuentran.

Un especial modelo de código de kernel '`-mcmodel=kernel`' es facilitado para código a nivel de sistema, tales como el kernel Linux. Un importante punto a observar es que por defecto en los AMD64 hay un área de 128-bytes de memoria ubicada por debajo del puntero de pila de los datos temporales, referida como "zona roja", la cual no es soportada por el kernel Linux. La compilación del kernel Linux en un AMD64 requiere la opción '`-mcmmodel=kernel -mno-red-zone`'.

8.2 Opciones para DEC Alpha

El procesador DEC Alpha tiene por defecto una configuración que maximiza el rendimiento de la aritmética en coma flotante y el coste de un soporte completo para las características de la aritmética del IEEE.

En la configuración por defecto en un procesador DEC Alpha no está habilitado el soporte para el tratamiento del infinito y gradual desbordamiento por defecto (números no normalizados). Las operaciones que producen el valor infinito o desbordamientos por defecto generarán excepciones en coma flotante (también conocidas como *traps*), y causarán que el programa termine, a menos que el sistema operativo capture y maneje las excepciones (lo cual es, en

[3] Intel ha añadido soporte para este conjunto de instrucciones como "Mejoras Intel 64 bits" a sus CPUs Xeon.

general, ineficiente). El estándar IEEE especifica que estas operaciones deben producir un resultado especial para representar las cantidades en el formato numérico del IEEE.

En la mayoría de los casos el comportamiento del DEC Alpha es aceptable, ya que la mayoría de los programas no producen ni el valor infinito ni desbordamientos por defectos. Para aplicaciones que requieran estas características, GCC facilita la opción '`-mieee`' para habilitar un completo soporte para la aritmética del IEEE.

Para demostrar la diferencia entre los dos casos el siguiente programa divide 1 entre 0:

```
#include <stdio.h>

int
main (void)
{
  double x = 1.0, y = 0.0;
  printf ("x/y = %g\n", x / y);
  return 0;
}
```

En aritmética del IEEE el resultado de 1/0 es `inf` (*Infinito*). Si el programa es compilado para un procesador Alpha con la configuración por defecto generará una excepción, que terminará el programa:

```
$ gcc -Wall alpha.es.c
$ ./a.out
Floating point exception     (en un procesador Alpha)
```

Usando la opción '`-mieee`' se asegura una completa conformidad con el IEEE —La división 1/0 produce correctamente el resultado `inf` y el programa continua ejecutándose correctamente:

```
$ gcc -Wall -mieee alpha.es.c
$ ./a.out
x/y = inf
```

Observe que los programas que generan excepciones en coma flotante corren más lentos cuando son compilados con la opción '`-mieee`', porque las excepciones son manejadas por software en vez de por hardware.

8.3 Opciones para SPARC

En el rango de procesadores SPARC la opción '`-mcpu=`*`CPU`*' genera código específico del procesador. Las opciones válidas para *`CPU`* son `v7`, `v8` (SuperSPARC), `Sparclite`, `Sparclet` y `v9` (UltraSPARC). El código producido con una opción específica no ejecutará en otros procesadores de la familia SPARC, excepto cuando exista una compatibilidad hacia atrás en el mismo procesador.

En sistemas UltraSPARC de 64 bits las opciones '`-m32`' y '`-m64`' controlan la generación de código para entornos de 32 ó 64 bits. El entorno de 32 bits seleccionado por '`-m32`' usa `int`, `long` y tipos de puntero con un tamaño de 32 bits. El entorno de 64 bits seleccionado con '`-m64`' usa un tipo `int` de 32 bits y `long` y tipos de puntero de 64 bits.

8.4 Opciones para POWER/PowerPC

En sistemas que usan la familia de procesadores POWER/PowerPC la opción '`-mcpu=`*`CPU`*' selecciona la generación de código específico para los modelos de CPU. Los posibles valores de *`CPU`* incluyen '`power`', '`power2`', '`powerpc`', '`powerpc64`' y '`common`', además de otros números de modelos específicos. El código generado con la opción '`-mcpu=common`' se ejecutará en cualquiera de los procesadores. La opción '`-maltivec`' habilita el uso del vector Altivec de procesamiento de instrucciones, si el soporte del hardware apropiado está disponible.

Los procesadores POWER/PowerPC incluyen una instrucción combinada "multiplicación y suma" $a * x + b$, la cual realiza las dos operaciones simultáneamente para ganar en velocidad —esto es referido como una *fusión* de multiplicaciones y sumas, y es usado por defecto en GCC. Debido a las diferentes formas en que los valores intermedios son redondeados, el resultado de una instrucción fundida puede no tener exactamente igual rendimiento que dos operaciones separadas. En los casos en que es requerida estrictamente la aritmética IEEE, el uso de las instrucciones combinadas puede ser deshabilitado con la opción '`-mno-fuser-madd`'.

En sistemas AIX, la opción '`-mminimal-toc`' disminuye el número de entradas GCC puestas en la *tabla de contextos* global (TOC) de los ejecutables para anular los errores "TOC overflow" en tiempo de enlazado. La opción '`-mxl-call`' realiza el enlazado de archivos objetos de GCC compatible con los compiladores de IBM's XL. Para aplicaciones que usan hilos POSIX, AIX siempre requiere la opción '`-pthread`' en compilación, incluso cuando el programa sólo se ejecute en modo hilo simple.

8.5 Soporte de múltiples arquitecturas

Un número de plataformas puede ejecutar código para más de una arquitectura. Por ejemplo, plataformas de 64 bits como AMD64, MIPS64, Sparc64 y PowerPC64 soportan la ejecución de código tanto de 32 como de 64 bits. De forma análoga, los procesadores ARM soportan tanto código de ARM como un código más compacto denominado "Thumb". GCC puede ser construido para soportar múltiples arquitecturas en estas plataformas. Por defecto, el compilador generará archivos objetos de 64 bits, pero dando la opción '`-m32`' generará archivos objetos de 32 bits para la correspondiente arquitectura.[4]

Observe que el soporte de múltiples plataformas depende de la disponibilidad de las librerías correspondientes. En plataformas de 64 bits que soportan ejecutables tanto de 64 como 32 bits, las librerías de 64 bits son a menudo puestas en directorios '`lib64`' en vez de en directorios '`lib`', p.e. en '`/usr/lib64`' y '`/lib64`'. Las librerías de 32 bits se encuentran por defecto en directorios '`lib`' de otras plataformas. Esto permite que las librerías tanto de 32 bits como de 64 bits puedan existir con el mismo nombre y en el mismo sistema. Otros sistemas, como el IA64/Itanium, emplea los directorios '`/usr/lib`' y '`/lib`' para las librerías de 64 bits. GCC conoce estos caminos y los emplea apropiadamente cuando compila código para 64 bits ó 32 bits.

8.6 Usos de coma flotante

El estándar IEEE-754 define el comportamiento a nivel de bit de las operaciones aritméticas en coma flotante en todos los procesadores modernos. Esto permite que los programas numéricos sean portados entre diferentes plataformas con idénticos resultados, en principio. En la práctica, hay a menudo pequeñas variaciones causadas por las diferencias en el orden de las operaciones (dependiendo de los niveles de compilación y optimización) pero generalemnte son insignificantes.

Sin embargo, se pueden apreciar más notables discrepancias cuando se portan programas numéricos entre sitemas x86 y otras plataformas, porque la unidad de coma flotante x87 (FPU) en los procesadores x86 procesan los resultados empleando internamente precisión extendida (los valores serán convertidos a doble precisión sólo cuando son almacenados en memoria). En contraste, procesadores como SPARC, PA-RISC, Alpha, MIPS y

[4] Las opciones '`-maix64`' y '`-maix32`' son empleadas en AIX.

POWER/PowerPC funcionan con valores en doble precisión nativos.[5] Las diferencias entre estas implementaciones provocan cambiar el comportamiento en el redondeo y en undeflow/overflow, porque los valores intermedios tienen mayor precisión relativa y rango de exponentes cuando son procesados en precisión extendida.[6] En particular, las comparaciones que afectan a valores en precisión extendida pueden fallar donde los valores en doble precisión equivalente debería comparar igual.

Para evitar esta incompatibilidad, la unidad de coma flotante (FPU) ofrece a menudo un modo del hardware redondeando la doble precisión. En este modo el resultado de cada operación en coma flotante con precisión extendida es redondeada a doble precisión en registros en coma flotante por la FPU. Es importante observar que el redondeo sólo afecta a la precisión, y no al rango del exponente, así el resultado es un formato híbrido de doble precisión con un rango extendido de exponentes.

En sistemas BSD tales como FreeBSD, NetBSD y OpenBSD, el modo hardware de redondeo de doble precisión está por defecto. Alcanzado la mayor compatibilidad con plataformas con doble precisión nativa. En sistemas x86 GNU/Linux el modo por defecto es precisión extendida (con la aspiración de suministrar un incremento en la precisión). Para habilitar el modo de redondeo de doble precisión es necesario anular la configuración por defecto en la base por procesos usando la instrucción de máquina FLDCW "floating-point load control-word".[7] Una simple función puede ser invocada para ejecutar estas instrucciones que se muestran a continuación. Se usa la palabra reservada de la extensión de GCC `asm` para insertar la instrucción específica en la salida del lenguaje ensamblador:

```
void
set_fpu (unsigned int mode)
{
  asm ("fldcw %0" :  :  "m" (*&mode));
}
```

[5] Los procesadores Motorola 68k también usan registros de precisión extendida, como el x86.

[6] Para tener cantidades en registros de precisión extendida x87 la precisión relativa es 5.42×10^{-20} y el rango del exponente es $10^{\pm 4932}$. Los valores estándar de doble precisión tienen una precisión relativa de 2.22×10^{-16} y un rango del exponente de $10^{\pm 308}$.

[7] El sistema operativo guarda y recupera la palabra de control cuando alterna entre procesos, así que cada proceso mantiene su propia configuración.

La apropiada configuración de `mode` para el redondeo en doble precisión es `0x27F`. El valor del modo también controla el comportamiento del manejo de la excepción de coma flotante y la dirección de redondeo (para más detalles vea el manual de referencia de los procesadores Intel y AMD).

En sistemas x86 GNU/Linux, la función anterior puede ser llamada al inicio de cualquier programa para deshabilitar precisión excesiva. Esto reproducirá el resultado de los procesadores de doble precisión nativa, en ausencia de desbordamientos por exceso y por defecto.

El siguiente programa demuestra los diferentes modos de redondeo:

```
#include <stdio.h>

void
set_fpu (unsigned int mode)
{
  asm ("fldcw %0" :  :  "m" (*&mode));
}

int
main (void)
{
  double a = 3.0, b = 7.0, c;
#ifdef DOUBLE       /* Activacion de uso */
  set_fpu (0x27F);  /* de redondeo en */
#endif              /* doble precision */
  c = a / b;

  if (c == a / b) {
    printf ("comparacion exitosa\n");
  } else {
    printf ("resultado inesperado\n");
  }
  return 0;
}
```

En sistemas x86 GNU/Linux la comparación `c == a / b` puede producir un resultado inesperado si `c` es tomado desde memoria (doble precisión) mientras que `a / b` es procesado en precisión extendida, porque la fracción 3/7 tiene diferentes representaciones en precisión doble y extendida.

```
$ gcc -Wall fptest2.c
$ ./a.out
unexpecedt result
```

Configurando el modo de redondeo hardware a doble precisión previene que lo anterior suceda:

```
$ gcc -Wall -DDOUBLE fptest2.c
$ ./a.out
comparison succeeds
```

Observe que la palabra de control de la coma flotante afecta al entorno completo del proceso, incluyendo las funciones de librerías C que sean invocadas. Una consecuencia de esto es que la aritmética en coma flotante es reducida eficientemente a doble precisión, desde que se confía en operaciones de precisión extendida.

La palabra de control de la coma flotante sólo afecta al comportamiento de FPU x87. Las operaciones en coma flotante procesadas con instrucciones de SSE y SSE2 son siempre llevadas a cabo en doble precisión nativa. Así, las opciones combinadas

```
$ gcc -Wall -msse2 -mfpmath=sse  ...
```

son a menudo suficientes para eliminar los efectos de la precisión extendida. Sin embargo, algunas operaciones (tales como funciones transcendentales) no están disponibles en las extensiones SSE/SSE2 y serán procesadas por el FPU x87.

8.7 Portabilidad de los tipos con signo y sin signo

Los estándares C y C++ permiten que el tipo carácter `char` sea con signo o sin signo, dependiendo de la plataforma y el compilador. La mayoría de los sistemas, incluidos x86 GNU/Linux y Microsoft Windows, usan `char` con signo, pero aquellos basados en PowerPC y procesadores ARM usan habitualmente `char` sin signo.[8] Esto puede causar resultados inexperados cuando se portan programas entre plataformas que tienen diferentes tipos `char` por defecto.

El siguiente código demuestra la diferencia entre plataformas con tipos `char` con signo y sin signo:

```
#include <stdio.h>

int
main (void)
```

[8] MacOS X (Drawin) en PowerPC usa `char` con signo, por consistencia con otras arquitecturas Darwin.

```
{
  char c = 255;
  if (c > 128) {
    printf ("el caracter es sin signo (c = %d)\n", c);

  } else {
    printf ("el caracter es con signo (c = %d)\n", c);
  }
  return 0;
}
```

Con un `char` sin signo, la variable `c` toma el valor 255, pero con un `char` con signo viene a ser -1.

La forma correcta de manipular variables `char` en C es a través de las funciones portables declaradas en '`ctype.h`', tales como `isalpha`, `isdigit` y `isblank`, en vez de sus valores numéricos. El comportamiento de las expresiones condicionales no portables tales como `c > 'a'` depende del signado del tipo `char`. Si se requiere explícitamente una versión con signo o sin signo en cierto punto de un programa, se puede especificar usando `signed char` o `unsigned char`.

For existing programs which assume that `char` is signed or unsigned, GCC provides the options '`-fsigned-char`' and '`-funsigned-char`' to set the default type of `char`. Using these options, the example code above compiles cleanly with '`-Wall -W`' when `char` is unsigned:

```
$ gcc -Wall -W -funsigned-char signed.c
$ ./a.out
char is unsigned (c = 255)
```

However, when `char` is signed the value 255 wraps around to -1, giving a warning when compiled with '`-Wall -W`':

```
$ gcc -Wall -W -fsigned-char signed.c
signed.c: In function ‘main’:
signed.c:7: warning: comparison is always false due to
  limited range of data type
$ ./a.out
char is signed (c = -1)
```

El mensaje de aviso *"comparison es always true/false due to limited range of data type"* es un síntoma de que el código asume una definición de `char` que difiere del tipo actual.

El problema más común en la escritura de código asumiendo el tipo `char` con signo ocurre con las funciones `getc`, `fgetc` y `getchar` (las cuales leen caractéres de un archivo). Tienen un tipo de retorno

de `int`, no `char`, y esto permite el uso de un valor especial -1 (definido como `EOF`) para indicar un error de final de archivo. Desafortunadamente, muchos programas han sido escritos guardando incorrectamente este valor de retorno directamente en una variable `char`. Aquí está el típico ejemplo:

```
#include <stdio.h>

int
main (void)
{
  char c;
  while ((c = getchar()) != EOF) /* no portable */
    {
      printf ("leer c = '%c'\n", c);
    }
  return 0;
}
```

Esto sólo funciona en plataformas que por defecto tienen el tipo `char` con signo.[9] En plataformas que usan `char` sin signo el mismo código puede fallar, porque el valor -1 se convierte en un 255 cuando se almacena en un `unsigned char`. Esto es normalmente causa de bucles infinitos porque el final del archivo no puede ser reconocido.[10] Para ser portable, el programa debe comprobar el valor de retorno como un entero antes de forzarlo a `char`, como a continuación:

```
#include <stdio.h>

int
main (void)
{
  int i;
  while ((i = getchar()) != EOF)
    {
      unsigned char c = i;
      printf ("leer c = '%c'\n", c);
    }
  return 0;
}
```

[9] Esto es además un sutil error en plataformas con `char` con signo —el carácter ASCII 255 es falsamente interpretado como una condición de final de archivo.

[10] Si se mostrase, el carácter con código 255 a menudo aparece como ´"y.

Las mismas consideraciones de esta sección se aplican a las definiciones de campos orientados a bit en estructuras, las cuales pueden ser por defecto con signo o sin signo. En GCC, el tipo por defecto para los campos orientados a bit puede ser controlado usando las opciones '`-fsigned-bitfields`' y '`-funsigned-bitfields`'.

9 Resolución de problemas

GCC proporciona varias opciones de ayuda para ayudar en la resolución de problemas con el proceso de compilación. Todas las opciones descritas en este capítulo funciona tanto con `gcc` como con `g++`.

9.1 Opciones de ayuda en línea de comandos

Para tener un breve recordatorio de varias opciones del comando, GCC tiene una opción de ayuda que muestra un resumen de alto nivel de las opciones del comando GCC:

```
$ gcc --help
```

Para mostrar una lista completa de opciones para `gcc` y sus programas asociados, tales como el Enlazador de GNU y el Ensamblador de GNU, usa la opción de arriba con la opción verbosa (‘`-v`’:

```
$ gcc -v --help
```

La lista completa de opciones producidas por este comando es extremadamente larga —se puede desear paginar a través de ésta usando el comando `more`, o redirigir la salida a un fichero por referencia:

```
$ gcc -v --help 2>&1 | more
```

9.2 Números de versión

Se puede encontrar el número de versión de `gcc` usando la opción de versión

```
$ gcc --version
gcc (Debian 4.6.1-4) 4.6.1
```

El número de versión es importante cuando se investigan los problemas de compilación debido a que las versiones antiguas de GCC pueden no tener algunas funcionalidades que un programa usa. El número de versión tiene las forma *major-version.minor-version* ó *major-version.minor-version.micro-version*, dónde el tercer número de versión “micro” (como se muestra arriba) es usado para la subsiguiente entrega de corrección de errores en una serie de entregas.

Pueden encontrarse más detalles acerca de la versión usando ‘`-v`’:

```
$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/i386-linux-gnu/gcc/
  /i486-linux-gnu/4.6.1/lto-wrapper
Target: i486-linux-gnu
Configured with: ../src/configure -v
--with-pkgversion='Debian 4.6.1-4'
--with-bugurl=file:///usr/share/doc/gcc-4.6/
  /README.Bugs
--enable-languages=c,c++,fortran,objc,obj-c++,go
--prefix=/usr --program-suffix=-4.6
--enable-shared --enable-multiarch
--with-multiarch-defaults=i386-linux-gnu
--enable-linker-build-id --with-system-zlib
--libexecdir=/usr/lib/i386-linux-gnu
--without-included-gettext --enable-threads=posix
--with-gxx-include-dir=/usr/include/c++/4.6
--libdir=/usr/lib/i386-linux-gnu --enable-nls
--enable-clocale=gnu --enable-libstdcxx-debug
--enable-libstdcxx-time=yes --enable-plugin
--enable-objc-gc --enable-targets=all
--with-arch-32=i586 --with-tune=generic
--enable-checking=release --build=i486-linux-gnu
--host=i486-linux-gnu --target=i486-linux-gnu
Thread model: posix
gcc version 4.6.1 (Debian 4.6.1-4)
```

Incluye información en los flags de construcción al mismo compilador y al archivo de configuración instalado, '`specs`'.

9.3 Compilación verbosa

La opción '`-v`' también puede ser usada para mostrar información detallada acerca de la secuencia exacta de comandos usados para compilar y enlazar un programa. Aquí hay un ejemplo que muestra la compilación verbosa del programa *Hola Mundo*:

```
$ gcc -v -Wall hola.es.c Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/lib/i386-linux-gnu/gcc/
  /i486-linux-gnu/4.6.1/lto-wrapper
Target: i486-linux-gnu
Configured with: ../src/configure
-v --with-pkgversion='Debian 4.6.1-4'
```

```
--with-bugurl=file:///usr/share/doc/gcc-4.6/
  /README.Bugs
--enable-la nguages=c,c++,fortran,objc,obj-c++,go
--prefix=/usr --program-suffix=-4.6 --enable-shared
--enable-multiarch
--with-multiarch-defaults=i386-linux-gnu
--enable-linker-build-id --with-system-zlib
--libexecdir=/usr/lib/i386-linux-gnu
--without-included-gettext --enable-threads=posix
--with-gxx-include-dir=/usr/include/c++/4.6
--libdir=/usr/lib/i386-linux-gnu --enable-nls
--enable-clocale=gnu --enable-libstdcxx-debug
--enable-libstdcxx-time=yes --enable-plugin
--enable-objc-gc --enable-targets=all
--with-arch-32=i586 --with-tune=generic
--enable-checking =release --build=i486-linux-gnu
--host=i486-linux-gnu --target=i486-linux-gnu
Thread model: posix gcc version 4.6.1
  (Debian 4.6.1-4)
COLLECT_GCC_OPTIONS='-v' '-Wall' '-mtune=generic'
'-march=i586'
/usr/lib/i386-linux-gnu/gcc/i486-linux-gnu/4.6.1/cc1
-quiet -v hola.es.c -quiet -dumpbase hola.es.c
-mtune=generic -march=i586 -auxbase hola.es -Wall -version
-o /tmp/cchpM0t3.s
GNU C (Debian 4.6.1-4) version 4.6.1
(i486-linux-gnu) compiled by GNU C version 4.6.1, GMP
version 5.0.1, MPFR version 3.0.1-p3, MPC version 0.9
warning: GMP header version 5.0.1 differs from library
version 5.0.2.
GGC heuristics: --param ggc-min-expand=46
--param ggc-min-heapsize=31802
ignoring nonexistent directory
"/usr/local/include/i386-linux-gnu"
ignoring nonexistent directory
"/usr/lib/i386-linux-gnu/gcc/i486-linux-gnu/4.6.1/../
  /../../../../i486-linux-gnu/include"
#include "..." search starts here:
#include <...> search starts here:
/usr/lib/i386-linux-gnu/gcc/i486-linux-gnu/4.6.1/include
/usr/local/include
/usr/lib/i386-linux-gnu/gcc/i486-linux-gnu/4.6.1/
  /include-fixed
```

```
/usr/include/i386-linux-gnu /usr/include
End of searchlist.
GNU C (Debian 4.6.1-4) version 4.6.1
(i486-linux-gnu) compiled by GNU C version 4.6.1, GMP
version 5.0.1, MPFR version 3.0.1-p3, MPC version 0.9
warning: GMP header version 5.0.1 differs from library
version 5.0.2.
GGC heuristics: --param ggc-min-expand=46
--param ggc-min-heapsize=31802
Compiler executable checksum:
2cfae8623c84fd817bfff483158c4341
COLLECT_GCC_OPTIONS='-v' '-Wall' '-mtune=generic'
'-march=i586' as --32 -o /tmp/ccQxW8a5.o /tmp/cchpM0t3.s
COMPILER_PATH=/usr/lib/i386-linux-gnu/gcc/
  /i486-linux-gnu/4.6.1/:
/usr/lib/i386-linux-gnu/gcc/i486-linux-gnu/4.6.1/:
/usr/lib/i386-linux-gnu/gcc/i486-linux-gnu/:
/usr/lib/i386-linux-gnu/gcc/i486-linux-gnu/4.6.1/:
/usr/lib/i386-linux-gnu/gcc/i486-linux-gnu/
LIBRARY_PATH=/usr/lib/i386-linux-gnu/gcc/
  /i486-linux-gnu/4.6.1/:
/usr/lib/i386-linux-gnu/gcc/i486-linux-gnu/4.6.1/
  /../../../:
/lib/:/usr/lib/:/usr/lib/i386-linux-gnu/
COLLECT_GCC_OPTIONS='-v' '-Wall' '-mtune=generic'
'-march=i586'
/usr/lib/i386-linux-gnu/gcc/i486-linux-gnu/4.6.1/
  /collect2
--build-id --no-add-needed --eh-frame-hdr -m elf_i386
--hash-style=both -dynamic -linker /lib/ld-linux.so.2
/usr/lib/i386-linux-gnu/gcc/i486-linux-gnu/4.6.1/
  /../../../crt1.o
/usr/lib/i386-linux-gnu/gcc/i486-linux-gnu/4.6.1/
  /../../../crti.o
/usr/lib/i386-linux-gnu/gcc/i486-linux-gnu/4.6.1/
  /crtbegin.o
-L/usr/lib/i386-linux-gnu/gcc/i486-linux-gnu/4.6.1
-L/usr/lib/i386-linux-gnu/gcc/i486-linux-gnu/4.6.1/
  /../../..
-L/usr/lib/i386-linux-gnu/tmp/ccQxW8a5.o
-lgcc --as-needed -lgcc_s --no-as-needed
-lc -l gcc --as-needed -lgcc_s --no-as-needed
/usr/lib/i386-linux-gnu/gcc/i486-linux-gnu/4.6.1/
```

```
  /crtend.o
/usr/lib/i386-linux-gnu/gcc/i486-linux-gnu/4.6.1/
  /../../../crtn.o
```

La salida producida por '`-v`' puede ser útil siempre y cuando haya un problema con el proceso de compilación en sí. Esto muestra las rutas completas usadas para buscar los ficheros de cabecera y librerías, los símbolos de preprocesador predefinidos, y los ficheros objeto y librerías usadas para enlazar.

9.4 Parando un programa en un bucle infinito

Un programa que se mete en un bucle infinito o se "cuelga" puede ser difícil de depurar. En la mayoría de los sistemas un proceso en primer plano puede ser detenido pulsando *`Control-C`*, que envía una señal de interrupción (SIGINT). Sin embargo, esto no ayuda en la depuración del problema —la señal SIGINT termina el proceso sin producir un core dump. Un enfoque más sofisticado es *adjuntar* el proceso en ejecución a un depurador e inspeccionarlo de manera interactiva.

Por ejemplo, aquí hay un programa simple con un bucle infinito:

```
int
main (void)
{
  usigned int i = 0;
  while (1) { i++; };
  return 0;
}
```

Para adjuntar el programa y depurarlo, el código debería ser compilado con la opción de depuración '`-g`':

```
$ gcc -Wall -g loop.es.c
$ ./a.out
(program hangs)
```

Una vez el ejecutable está ejecutándose, se necesitará encontrar su identificador de proceso (PID). Esto se puede hacer desde otra sesión con el comando `ps x`:

```
$ ps x
PID   TTY    STAT TIME COMMAND
...   .....    ..  ....
2333 pts/0    R+  0:16 ./a.out
```

En este caso el identificador de proceso es 2333, y ahora se puede adjuntar con `gdb`. El depurador debería ser iniciado en el directorio que contiene el ejecutable y su código fuente:[1]

```
$ gdb a.out
GNU gdb (GDB) 7.3-debian
...
(gdb) attach 2333
Attaching to program: a.out, process 2333
Reading symbols from
/lib/i386-linux-gnu/i686/cmov/libc.so.6...(no
debugging symbols found)...done.
Loaded symbols for
/lib/i386-linux-gnu/i686/cmov/libc.so.6
Reading symbols from /lib/ld-linux.so.2...(no debugging
symbols found)...done.
Loaded symbols for /lib/ld-linux.so.2
0x080483c5 in main () at loop.c:5
5         while (1) { i++; };
(gdb)
```

La salida muestra la línea que se ejecutó en el punto en el que el depurador se adjuntó al proceso. El programa adjunto se pausa pero todavía 'vive' —este puede examinado interactivamente y continúa o termina (con el comando `kill`) si es necesario:

```
(gdb) print i
$1 = 1960534048
(gdb) kill
Kill the program being debugged? (y or n) y
(gdb)
```

Si se quiere parar un proceso inmediatamente y crear un core dump, el comando de shell `kill -3` *`pid`* (dónde *pid* es el identificador de proceso) enviará una señal SIGQUIT. La señal SIGQUIT dispara un core dump, no como SIGINT. Nótese que si los core dump fueran deshabilitados cuando el proceso fué iniciado, ningún fichero core se producirá (véase Sección 5.1 [Examinando archivos core], página 49).

[1] De manera alternativa, las rutas apropiadas pueden ser configuradas en `gdb` usando los comandos `file` y `directory`.

9.5 Previniendo un uso excesivo de memoria

Algunas veces un error de programación hará que un proceso gestione grandes cantidades de memoria, consumiendo toda la RAM en un sistema. Para prevenir esto, el comando GNU Bash `ulimit -v` *`limit`* se puede usar para restringir la cantidad de memoria virtual disponible en cada proceso. El límite es medido en kilobytes y se aplica a nuevos procesos iniciados en la shell actual. Por ejemplo,

```
$ ulimit -v 4096
```

limitará los siguientes procesos a 4 megabytes de memoria virtual (4096k). Por defecto el límite no puede incrementarse en la misma sesión una vez ha sido establecido, así es mejor comenzar una shell distinta para operaciones `ulimit` reducidas. De manera alternativa, se puede establecer un *límite blando* (que puede ser deshecho) con las opciones '`-S -v`'.

Adicionalmente para prevenir derroche de procesos, limitando la cantidad de memoria que un programa puede reservar permite una forma de comprobar la robusted del manejo de condiciones *out of memory*. Un artificial límite bajo se puede usar para simular ejecuciones fuera de memoria —un programa bien escrito no debe fallar en estos casos.

El comando `ulimit` soporta otras opciones incluyendo '`-p`', que restringe el número de procesos hijos que pueden ser creados, y '`-t`', que establece un límite en el número de segundos de CPU en el que un proceso puede ejecutarse. La lista completa de configuraciones puede mostrarse con el comando `ulimit -a`. Para mostrar más información acerca del comando `ulimit`, escribe `help ulimit` en una Shell de Bash.

10 Utilidades relativas al compilador

Este capítulo describe un conjunto de herramientas que son muy utilizadas junto a GCC. Estas incluyen el archivador GNU `ar`, para crear librerías, y los programas de comprobación de perfil y cobertura, `gprof` y `gcov`.

10.1 Creando una librería con el archivador de GNU

El archivador GNU `ar` combina una colección de archivos objeto en un simple fichero de archivo, conocido también como una *librería*. Un fichero de archivo es simplemente una forma conveniente de distribuir un grán número de archivos objeto juntos (como se describió anteriormente en Sección 2.7 [Enlazando con librerías externas], página 18).

Para demostrar el uso del archivador GNU se creará una pequeña librería '`libhola.a`' conteniendo dos funciones `hola` y `adios`.

El primer archivo objeto será generado desde el código fuente de la función `hola`, en el archivo '`hola_fn.es.c`':

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

void
hola (const char * nombre)
{
  printf ("!Hola, %s!\n", nombre);
}
```

El segundo archivo objeto será generado con el archivo fuente '`adios_fn.es.c`', que contiene la función `adios`:

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

void
bye (void)
{
  printf ("!Adios!\n");
}
```

Ambas funciones usan el archivo de cabecera '`hola.es.h`', ahora con un prototipo para la función `adios()`:

```
void hola (const char * nombre);
void adios (void);
```

El código fuente puede ser compilado a los archivos objeto '`hola_fn.o`' y '`adios_fn.o`' usando el comando:

```
$ gcc -Wall -c hola_fn.es.c
$ gcc -Wall -c adios_fn.es.c
```

Estos archivos objeto pueden ser combinados en una librería usando la siguiente línea de comando:

```
$ ar cr libhola.a hola_fn.o adios_fn.o
```

La opción '`cr`' significa "crear y reemplazar".[1] Si la librería no existe, primero será creada. Si la librearía ya existe, el archivo original con el mismo nombre será reemplazado por el nuevo archivo especificado en la línea de comando. El primer argumento '`libhola.a`' es el nombre de la librería. El resto de argumentos son los nombres de los archivos objeto que serán copiados a la librería.

El archivador `ar` además suministra una opción de "tabla de contenidos" '`t`' para listar los archivos objeto de una libraría existente:

```
$ at t libhola.a
hola_fn.o
adios_fn.o
```

Observe que cuando la librería sea distribuida, los archivos cabecera de las funciones y variables públicas será suministrado para que también esté disponible, así el usuario final puede incluirlos y obtener los prototipos correctos.

Se puede escribir ahora un programa usando las funciones de la recien creada librería:

```
#include "hola.h"

int
main (void)
{
  hola ("cualquiera");
  adios ();
  return 0;
}
```

Este archivo puede ser compilado con la siguiente línea de comando, como está descrito en Sección 2.7 [Enlazando con librerías externas], página 18, asumiendo que la librería '`libhola.a`' está almacenada en el directorio actual:

[1] Observe que `ar` no requiere el prefijo '`-`' para estas opciones.

```
$ gcc -Wall main3.c libhello.a -o hello
```

El programa principal es enlazado junto a los archivos objeto encontrados en el archivo de librería '`libhola.a`' para producir el ejecutable final.

La opción de atajo de enlazado de la librería se puede usar para enlazar el programa, sin necesidad de especificar explícitamente un camino completo a la librería:

```
$ gcc -Wall -L. main3.c -lhello -o hello
```

La opción '`-L.`' es necesaria para añadir el directorio actual al camino de búsqueda de librerías. El ejecutable resultante se puede ejecutar como habitualmente:

```
$ ./hola
¡Hola, cualquiera!
¡Adios!
```

Se muestra la salida de ambas funciones `hola` y `adios` definidas en la librería.

10.2 Usando el profiler `gprof`

El GNU profiler `gprof` es una muy usada herramienta para medir el rendimiento de un programa —graba el número de llamadas de cada función y contabiliza el tiempo que emplea en ello, en una base por función. Las funciones que consumen largas fracciones del tiempo de ejecución pueden ser identificas fácilmente a partir de la salida de `gprof`. Mejorar la velocidad de un programa requiere primero concentración sobre aquellas funciones que dominan el tiempo de ejecución completamente.

Se usará `gprof` para examinar el rendimiento de un pequeño programa numérico que procesará una larga secuencia irresuelta *Collatz conjeture* en matemáticas.[2] La conjetura de Gollatz afecta a secuencias definidas por la regla:

$$x_{n+1} \leftarrow \begin{cases} x_n/2 & \text{si } x_n \text{ es par} \\ 3x_n + 1 & \text{si } x_n \text{ es impar} \end{cases}$$

La secuencia es iterada desde el valor inicial x_0 hasta que termina con el valor 1. De acuerdo con la conjetura, todas las secuencias terminarán eventualmente —el programa mostrará las largas secuencias como incrementos de x_0. El archivo fuente '`collatz.es.c`' contiene tres funciones: `main`, `nseq` y `step`:

[2] American Mathematical Monthly, Volume 92 (1985), 3–23

```
#include <stdio.h>

/* Calcula el tamano de secuencias Collatz */

unsigned int
step (unsigned int x)
{
  if (x % 2 == 0)
    {
      return (x / 2);
    }
  else
    {
      return (3 * x + 1);
    }
}

unsigned int
nseq (unsigned int x0)
{
  unsigned int i = 1, x;

  if (x0 == 1 || x0 == 0)
    return i;

  x = step (x0);

  while (x != 1 && x != 0)
    {
      x = step (x);
      i++;
    }

  return i;
}

int
main (void)
{
  unsigned int i, m = 0, im = 0;

  for (i = 1; i < 500000; i++)
    {
      unsigned int k = nseq (i);

      if (k > m)
        {
          m = k;
          im = i;
```

```
            printf ("tamano de secuencia = %u para %u\n", m, im);
          }
      }

    return 0;
}
```

Para usar profiling, el programa debe ser compilado y enlazado con la opción de profiling '`-pg`':

```
$ gcc -Wall -c -pg collatz.es.c
$ gcc -Wall -pg collatz.o
```

Esto creará un ejecutable *instrumented* que contiene instrucciones adicionales para registrar el tiempo consumido en cada función.

Si el programa consiste en más de un archivo fuente la opción '`-pg`' se debe usar cuando se compila cada archivo fuente, y será usado en el enlazado de los archivos objetos para crear el ejecutable final (como se muestra anteriormente). Es un error común olvidar la opción '`-pg`' en el enlazado, lo cual evita que el profiling registre mucha información útil.

El ejecutable se debe ejecutar para obtener los datos del profiling:

```
$ ./a.out
```

(la salida normal del programa es mostrada)

Mientras está corriendo el ejecutable instrumentado, datos de profiling son silenciosamente escritos en el archivo '`gmon.out`' en el directorio actual. Puede se analizado con `gprof` dando el nombre del ejecutable como argumento:

```
$ gprof a.out
Flat profile:
Each sample counts as 0.01 seconds.
 %     cumul.    self              self   total
time  seconds seconds    calls us/call us/call name
68.59    2.14    2.14 62135400    0.03    0.03 step
31.09    3.11    0.97   499999    1.94    6.22 nseq
 0.32    3.12    0.01                          main
```

La primera columna de los datos muestra que el programa emplea la mayoría de su tiempo (casi del 70%) en la función `step`, y el 30% en `nseq`. Consecuentes esfuerzos para decrementar el tiempo de ejecución debe concentrarse en esta forma. En comparación, el tiempo empleado en la función `main` es completamente imperceptible (menos del 1%).

Las otras columnas de la salida suministran información del número total de llamadas que son hechas a las funciones, y el tiempo

gastado en cada función. Información adicional del tiempo de ejecución se muestra con `gprof` pero aquí no se muestra. Un completo detalle se puede encontrar en el manual "*GNU gprof —El GNU Profiler*", de Jay Fenlason y Richard Stallman.

10.3 Test de cobertura con `gcov`

La herramienta de test de cobertura GNU `gcov` analiza el número de veces que cada línea del programa es ejecutada. Esto permite encontrar áreas del código que no son usadas, o que no se ejecutan en los tests. Cuando se combinan la información de perfilación con la información de cobertura se logra aumentar la velocidad de un programa concentrándose en líneas específicas del código fuente.

Se usará el ejemplo siguiente para demostrar `gcov`. Este programa recorre los números del 1 al 9 y comprueba su divisibilidad con el operador módulo (`%`).

```
#include <stdio.h>

int
main (void)
{
  int i;

  for (i = 1; i < 10; i++)
    {
      if (i % 3 == 0)
        printf ("%d es divisible por 3\n", i);
      if (i % 11 == 0)
        printf ("%d es divisible por 11\n", i);
    }

  return 0;
}
```

Para habilitar los test de cobertura el programa debe ser compilado con las siguientes opciones:

```
$ gcc -Wall -fprofile-arcs -ftest-coverage cov.es.c
```

Esto creará un ejecutable instrumentado que contiene instrucciones adicionales que registran el número de veces que cada línea del programa es ejecutada. La opción '`-ftest-coverage`' añade instrucciones para contabilizar el número de veces que las líneas individuales son ejecutadas, mientras que '`-fprofile-arcs`' incorpora instrucciones de código por cada bifurcación del programa. Las instrucciones de bifurcación registran la frecuencia de los diferentes

caminos que toman las instrucciones '`if`' y otras condicionales. El ejecutable debe ser ejecutado para crear los datos de cobertura:

```
$ ./a.out
3 es divisible por 3
6 es divisible por 3
9 es divisible por 3
```

Los datos de la ejecución son escritos en varios archivos con extensiones '`.bb`', '`.bbg`' y '`.da`' respectivamente en el directorio actual. Estos datos pueden ser analizados usando el comando `gcov` y el nombre del archivo fuente:

```
$ gcov cov.es.c
 88.89% of 9 source lines executed in file cov.es.c
Creating cov.es.c.gcov
```

El comando `gcov` produce una anotación de la versión del archivo fuente original, con la extensión '`.gcov`', conteniendo los contadores de los números de veces que cada línea fué ejecutada:

```
           #include <stdio.h>

           int
           main (void)
           {
        1    int i;

       10    for (i = 1; i < 10; i++)
               {
        9        if (i % 3 == 0)
        3          printf ("%d is divisible by 3\n", i);
        9        if (i % 11 == 0)
   ######          printf ("%d is divisible by 11\n", i);
        9      }

        1    return 0;
        1  }
```

Los contadores de líneas pueden verse en la primera columna. Las líneas que no son ejecutadas están marcadas con almohadillas '`######`'. El comando '`grep '######' *.gcov`' se puede usar para encontrar las partes que no están siendo usadas.

11 Como funciona el compilador

Este capítulo describe en detalle cómo GCC transforma ficheros fuente a un fichero ejecutable. La compilación es un proceso multi-fase involucrando varias herramientas, incluyendo el Compilador de GNU en sí (a través de los frontends `gcc` o `g++`), el Ensamblador de GNU `as`, y el Enlazador de GNU `ld`. El conjunto completo de herramientas usadas en el proceso de compilación es llamado *toolchain*.

11.1 Una vista de los procesos de compilación

La secuencia de comandos ejecutados por una simple invocación de GCC consiste en las siguientes fases:

- preprocesamiento (expandir macros)
- compilación (desde código fuente a lenguaje ensamblador)
- ensamblaje (desde lenguaje ensamblador a código máquina)
- enlace (crear el ejecutable final)

Como ejemplo, se examinarán estas fases de compilación individualmente usando el programa *Hola Mundo* '`hola.es.c`':

```
#include <stdio.h>

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

Nótese que no es necesario usar cualquiera de los comandos individuales descritos en esta sección para compilar un programa. Todos los comandos son ejecutados de manera automática y transparente por GCC internamente, y puede ser visto usando la opción '`-v`' descrita antes (véase Sección 9.3 [Compilación verbosa], página 92). El propósito de este capítulo es proporcionar un entendimiento de cómo el compilador funciona.

Aunque el programa *Hola Mundo* es muy simple, éste usa cabeceras externas y librerías, y así ejercita todos los pasos importantes del proceso de compilación.

11.2 El preprocesador

La primera fase del proceso de compilación es el uso del preprocesador para expandir macros y ficheros de cabecera incluidos. Para realizar esta fase, GCC ejecuta el siguiente comando:[1]

```
$ cpp hola.es.c > hola.i
```

El resultado es un fichero '`hola.i`' que contiene el código fuente con todas las macros expandidas. Por convención, ficheros preprocesados son dados por la extensión de fichero '`.i`' para programas C y '`.ii`' para programas C++. En la práctica, el fichero preprocesado no es guardado en disco a menos que la opción '`-save-temps`' sea usada.

11.3 El compilador

La siguiente fase del proceso es compilación de código fuente preprocesado a lenguaje ensamblador, para un procesador específico. La opción '`-S`' dicta a `gcc` a convertir el código fuente C a lenguaje ensamblador sin crear un fichero objeto:

```
$ gcc -Wall -S hola.i
```

El lenguaje ensamblador resultante es almacenado en el fichero '`hola.s`'. Aquí está el lenguaje ensamblador de *Hola Mundo* para un procesador Intel x86 (i686):

```
$ cat hello.s
        .file   "hello.c"
        .section        .rodata
.LC0:
        .string "Hello, world!"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushl   %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl    %esp, %ebp
        .cfi_def_cfa_register 5
        andl    $-16, %esp
```

[1] Como se mencionó antes, el preprocesador está integrado en el compilador en versiones recientes de GCC. Conceptualmente, el proceso de compilación es el mismo que ejecutar el preprocesar como una aplicación separada.

```
        subl    $16, %esp
        movl    $.LC0, (%esp)
        call    puts
        movl    $0, %eax
        leave
        .cfi_restore 5
        .cfi_def_cfa 4, 4
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (Debian 4.6.1-4) 4.6.1"
        .section        .note.GNU-stack,"",@progbits
```

Nótese que el lenguaje ensamblador contiene una llamada a la función externa `puts`, una versión simple de `printf` para cadenas que no contienen caracteres formateados.

11.4 El ensamblador

El propósito de un ensamblador es convertir lenguaje ensamblador a código máquina y generar un fichero objeto. Cuando hay llamadas a funciones externas en el fichero fuente de ensamblador, el ensamblador deja las direcciones las funciones externas indefinidas, para ser rellenadas después por el enlazador. El ensamblador puede ser invocado con la siguiente línea de comandos:

```
$ as hola.s -o hola.o
```

As with GCC, the output file is specified with the '`-o`' option. The resulting file '`hello.o`' contains the machine instructions for the *Hello World* program, with an undefined reference to `puts`.

11.5 El enlazador

La fase final de compilación es el enlace de ficheros objeto para crear un ejecutable. En la práctica, un ejecutable requiere muchas funciones externas del sistema y de librerías C. Por consiguiente, los comandos de enlace actuales usados internamente por GCC son complicados. Por ejemplo, el comando completo para enlazar el programa *Hola Mundo* es:[2]

[2] El comando preciso varía dependiendo de la versión de `gcc` y el sistema operativo. Versiones recientes de `gcc` usan wrapper llamado `collect2` que hace algún trabajo adicional antes de llamar a `ld`.

```
$ ld -dynamic-linker /lib/ld-linux.so.2 \
                     /usr/lib/crt1.o \
                     /usr/lib/crti.o \
       /usr/lib/gcc/i486-linux-gnu/4.3/crtbegin.o \
     -L/usr/lib/gcc/i486-linux-gnu/4.3 hola.o \
     -lgcc -lgcc_eh \
     -lc /usr/lib/gcc/i486-linux-gnu/4.3/crtend.o \
     /usr/lib/crtn.o
```

Afortunadamente nunca se necesita escribir el comando de arriba directamente —el proceso entero de enlace es manejado de manera transparente por `gcc` cuando se invoca como sigue:

```
$ gcc hola.o
```

Esto enlaza el fichero objeto '`hola.o`' a la librería estándar de C, y produce un fichero ejecutable '`a.out`':

```
$ ./a.out
¡Hola, mundo!
```

Un fichero objeto para un programa C++ puede ser enlazado a la librería estándar C++ del mismo modo con un simple comando `g++`.

12 Examinado archivos compilados

Este capítulo describe varias herramientas comunes para examinar el contenido de los archivos objeto y ejecutables.

12.1 Identificando archivos

Cuando un archivo fuente ha sido compilado a un archivo objeto o a un ejecutable las opciones usadas en la compilación no son obvias. El comando `file` examina el contenido del archivo objeto o ejecutable y determina algunas de sus características, tal como si fué compilado para enlazado estático o dinánico.

Por ejemplo, aquí se muestra el resultado del comando `file` para un ejecutable típico:

```
$ file a.out
a.out: ELF 32-bit LSB executable, Intel 80386,
  version 1 (SYSV), dynamically linked (uses shared
  libs), not stripped
```

La salida muestra que el archivo ejecutable es enlazado con librerías compartidas dinámicamente, y compilado para procesadores Intel 386 y compatibles. Una completa explicación de la salida se muestra a continuación:

`ELF` El formato interno de un archivo ejecutable (ELF proviene de "Executable and Linking Format", otros formatos como el COFF "Common Object File Format" fueron usados en antiguos sistemas operativos (p.e. MS-DOS)).

`32 bits` El tamaño de palabra (para algunas plataformas esto sería 64 bits).

`LSB` Compilado para una plataforma con ordenación de palabras donde el primero es el byte menos significativo (LSB, *least significat byte*), tales como los procesadores Intel y AMD x86 (la alternativa el primero es el byte más significativo (MSB, *most significant byte*) es usada en otros procesadores, tales como Motorola 680x0)[1]. Algunos procesadores como Itanium y MIPS soportan ambas ordenaciones LSB y MSB.

[1] La ordenación MSB y LSB son conocidas como big-endian y little-endian respectivamente (el término original es de la sátira "Los viajes de Gulliver" de Jonathan Swift's, de 1727).

`Intel 80386`
: Procesador para el cual ha sido complilado el archivo ejecutable.

`version 1 (SYSV)`
: Esta es la versión del formato interno del archivo.

`dynamically linked`
: El ejecutable emplea librerías compartidas dinámicas (`statically linked` indicaría que el programa fué enlazado estáticamente, en el ejemplo usando la opción '`-static`')

`not stripped`
: El ejecutable contiene una tabla de símbolos (ésta puede ser eliminada usando el comando `strip`).

El comando `file` puede ser usado con archivos objeto, donde dará una salida similiar. El estándar POSIX[2] para sistemas Unix define el comportamiento del comando `file`.

12.2 Examinando la tabla de símbolos

Tal y como se describió anteriormente en la discusión sobre depuración, los archivos ejecutables y objeto pueden contener una tabla de símbolos (véase Capítulo 5 [Compilando para depuración], página 49). Esta tabla almacena la localización de las funciones y variables por nombre, y puede ser mostrada con el comando `nm`:

```
$ nm a.out
08049534 d _DYNAMIC
08049608 d _GLOBAL_OFFSET_TABLE_
........
080483e4 T main
         U puts@GLIBC_2.0
```

Entre el contenido de la tabla de símbolos, la salida muestra que el inicio de la función `main` está en el desplazamiento hexadecimal `080483e4`. La mayoría de símbolos es para uso interno del compilador y del sistema operativo. Una '`T`' en la segunda columna indica que esta función está definida en el archivo objeto, mientras que una '`U`' indica que esta función está indefinida (y se deberá resolver en el enlazado con otro archivo objeto). Una completa explicación de la salida de `nm` se puede encontrar en la manual *GNU Binutils*.

[2] POSIX.1 (2003 edition), IEEE Std 1003.1.2003.

El uso más frecuente del comando `nm` es para comprobar si una librería contiene la definición de una función específica, observando las entradas '`T`' en la segunda columna del nombre de la función.

12.3 Encontrando librerías dinámicas enlazadas

Cuando un programa ha sido compilado usando librerías compartidas necesita cargar estas librerías dinámicamente en tiempo de ejecución para realizar las llamadas a funciones externas. El comando `ldd` examina un ejecutable y muestra la lista de librerías compartidas que necesita. Estas librerías son referidas como librerías compartidas *dependientes* del ejecutable.

Por ejemplo, el siguiente comando muestra como se encuentra la librería compartida como dependencia del programa *Hola Mundo*:

```
$ gcc -Wall hola.es.c
$ ldd a.out
libc.so.6 => /lib/libc.so.6 (0x40020000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

La salida muestra que el programa *Hola Mundo* depende de la libraría C `libc` (librería compartida versión 6) y de la librería de carga dinámica `ld-linux` (librería compartida versión 2).

Si el programa usa librerías externas, tal como la librería matemática, éstas también serían mostradas. Por ejemplo, el programa `calc` (el cual usa la función `sqrt`) genera la siguiente salida:

```
$ gcc -Wall calc.es.c -lm -o calc
$ ldd calc
libm.so.6 => /lib/libm.so.6 (0x40020000)
libc.so.6 => /lib/libc.so.6 (0x40041000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

La primera línea muestra que este programa depende de la librería matemática `libm` (librería compartida verisón 6), además de las librería de C y la librería de carga dinámica.

El comando `ldd` puede ser usado para examinar estas mismas librerías compartidas, con objeto de seguir la cadena de dependencia de librerías compartidas.

13 Mensajes comunes de error

Este capítulo describe los errores y avisos más frecuentes producidos por `gcc` y `g++`. Cada caso está acompañado por una descripción de las causas, un ejemplo y sugerencias de posibles soluciones. Nótese que los mensajes de error puede variar ligeramente entre las diferentes versiones del compilador. Algunos mensajes puede solo se mostrados con opciones tales '`-Wall`' y '`-W`'.

13.1 Mensajes de error del preprocesador

`No such file or directory`

Este error ocurre si GCC no puede encontrar un fichero pedido en su ruta de búsqueda. El fichero puede haber sido especificado en la línea de comandos, o con una sentencia `#include`. Si el nombre del fichero ha sido tecleado incorrectamente o el directorio del fichero necesita ser añadido para la ruta include o la ruta de enlace.

Ejemplo:

```
#include <stdoi.h>  /* incorrecto */

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

El programa de arriba intenta incluir el fichero inexistente '`stdoi.h`' dando el error '`stdoi.h: No such file or directory`'. El nombre de fichero correcto es '`stdio.h`'.

`macro or '#include' recursion too deep`
`#include nested too deeply`

Este error ocurre si el preprocesador encuentra demasiadas directivas '`#include`' anidadas. Esto es normalmente causado por dos ó más ficheros intentando incluir otro, provocando una recursión infinita.

Ejemplo:

```
/* foo.h */
#include "bar.h"
...
```

```
/* bar.h */
#include "foo.h"
...
```

La solución a este problema es asegurar que los ficheros no se incluyen mutuamente, o usar 'guardas include' (véase Sección 7.4.2 [Proporcionando sus propias plantillas], página 74 por ejemplo).

`invalid preprocessing directive #...`

Este error indica que el preprocesador encontró un comando `#` irreconocible.

Ejemplo:

```
#if FOO
   int x = 1;
#elsif BAR   /* deberia ser #elif */
   int x = 2;
#else
   int x = 3;
#endif
```

La sintaxis de preprocesador requiere `#elif` para la condición "else if" en los bloques `#if`, en vez de `#elseif`. En el ejemplo de arriba ocurre un error de directiva inválida en el incorrecto uso de `#elseif`, pero solo cuando `FOO` está definido (en otros casos el preprocesador ignora cualquiera de la sentencia `#else`).

13.2 Mensajes de error del compilador

'*variable*' `undeclared (first use in this function)`

En C y C++ las variables deben ser declaradas antes de poder ser usadas. Este mensaje de error indica que el compilador ha encontrado un nombre de variable que no tiene su declaración correspondiente. Esto puede ser causado por una declaración perdida, o por un error de escritura en el nombre. Los nombres de variables son sensibles a mayúsculas, así `foo` y `Foo` representan diferentes variables. Para mantener breve la salida, sólo se informa del primer uso de una variable no declarada.

Ejemplo:

```
int
main (void)
{
  int i;
  j = 0;      /* no declarado */
  return j;
}
```

La variable `j` no está declarada y saltará el error `‘j’ undeclared`.

`parse error before ‘...’`
`expected ’;’ before ‘...’`
`syntax error`

Este mensaje de error ocurre cuando el compilador encuentra una entrada inesperada, p.e. secuencias de caracteres que no siguen la sintaxis del lenguaje. Este mensaje de error salta por la ausencia de paréntesis o llaves de cierre o punto y coma precediendo la línea de error, o una palabra reservada inválida.

Ejemplo:

```
#include <stdio.h>

int
main (void)
{
  printf ("!Hola ") /*falta punto y coma*/
  printf ("Mundo!\n");
  return 0;
}
```

Aquí hay una ausencia de punto y coma después de la primera llamada a `printf`, dando un `error de parseo`.

`parse error at end of input`
`expected declaration or statement at end of input`

Este error ocurre si el compilador encuentra un final de archivo inesperadamente, tal como cuando se analizan un número no balanceado de llaves de apertura y cierre. Es a menudo causado por la ausencia de un cierre de llaves en algún lugar.

Ejemplo:

```
#include <stdio.h>

int
main (void)
{
  if (1) {
    printf ("!Hola Mundo!\n");
    return 0;  /* no cierra llave */
}
```

Una llave de cierre adicional es necesaria en este programa para prevenir el error `expected declaration or statement at end of input`.

`warning: implicit declaration of function '...'`
`warning: incompatible implicit declaration of built-in function '...'`

Este aviso es generado cuando una función es usada sin un prototipo declarado. Esto puede ser causado por la falta de inclusión de una archivo de cabeceras, o si no olvidando suministrar un prototipo de función.

Ejemplo:

```
int
main (void)
{
  printf ("!Hola Mundo!\n"); /*hace falta*/
  return 0;          /*fichero de cabecera*/
}
```

El archivo de cabeceras del sistema '`stdio.h`' no está incluido, así que el prototipo de la función `printf` no ha sido declarado. El programa necesita una línea inicial `#include <stdio.h>`.

`unterminated string or character constant`
`missing terminating " character`

Este error es causado por unas comillas de apertura de una cadena o carácter que no tiene su correspondiente comillas de cierre. Las comillas deben aparecer en parejas coincidentes, como una comilla simple `'a'` para caracteres o dobles comillas `"aaa"` para cadenas.

Ejemplo:

```
#include <stdio.h>

int
main (void)
{        /* No cierra comillas */
  printf ("!Hola Mundo!\n);
  return 0;
}
```

Las dobles comillas de apertura de la cadena en este programa no tiene su correspondiente dobles comillas de cierre, así el compilador lee el resto del archivo como parte de la cadena.

`character constant too long`

En C y C++ los caracteres son escritos usando simples comillas, p.e. `'a'` da el código ASCII de la letra a (67), y `'\n'` da el código ASCII de nueva línea (10). Este error ocurre si unas comillas simples son usadas para encerrar más de un carácter.

Ejemplo:

```
#include <stdio.h>

int
main (void)
{        /* Uso de comillas correctas */
  printf ('!Hola Mundo!\n');
  return 0;
}
```

El programa anterior confunde las comillas simples y dobles. Una secuencia de caracteres se debe escribir con comillas dobles, p.e. `"¡Hola Mundo!"`. Este mismo problema ocurre en el siguiente programa C++,

```
#include <iostream>

int
main (void)
{
  std::cout << 'Hello World!\n';  // wrong quotes
  return 0;
}
```

Este error puede ocurrir si la barra de división y la barra invertida son confundidas en una secuencia de

escape, p.e. usando `'/n'` en vez de `'\n'`. La secuencia `/n` consiste en dos caracteres separados, '`/`' y '`n`'.

Observe que de acuerdo con el estándar C no hay límite en la longitud de una constante carácter, pero el valor de una constante carácter que contiene más de un carácter está definido en la implementación. Versiones más recientes de GCC soportan constantes de carácter multi-byte, y en vez de avisar de un error `multiple-character character constant`,será generado en este caso `warning: character constant too long for its type`.

`warning: initialization makes integer from pointer without a cast`

Este error indica el uso erróneo de un puntero en un contexto entero. Técnicamente, es posible convertir entre tipos punteros y enteros, pero es raramente necesario fuera de las aplicaciones a nivel de sistema. Más a menudo, este aviso es el resultado de usar un puntero sin desreferenciarlo (p.e. escribiendo `int i=p` en vez de `int i = *p`).

Este aviso puede ocurrir con tipos `char` y `char *`, desde que `char` es un tipo entero.

Ejemplo:

```
int
main (void)
{
  char c = "\n";  /* incorrecto */
  return 0;
}
```

La variable `c` tiene tipo `char`, mientras que la cadena `'\n'` se evalua como un puntero a `const char *` (una zona de 2 bytes de memoria contiene el valor ASCII del carácter nueva línea seguido de un byte a cero `'\0'`, con el cual la cadena está nul-terminada). El código ASCII para el carácter nueva línea se puede encontrar usando `char c = '\n';`

Errores similares pueden ocurrir con el uso incorrecto de la macro `NULL`,

```
#include <stdlib.h>

int
main (void)
{
  int i = NULL;  /* incorrecto */
  return 0;
}
```

En C, la macro `NULL` es definida como `((void *)0)` en '`stdio.h`' y solamente debería ser usada en un contexto de puntero.

`dereferencing pointer to incomplete type`

Este error ocurre cuando un programa intenta acceder a un elemento de una estructura a través de un puntero sin que la estructura haya sido declarada anteriormente. En C y C++ es posible declarar punteros a estructuras antes de la declaración de las estructuras, proporcionando los punteros que no son referenciados —Esto es conocido como *declaración avanzada* (forward declaration).

Ejemplo:

```
struct btree * data;

int
main (void)
{
  data->size = 0;  /* tipo incompleto */
  return 0;
}
```

Este programa tiene una declaración posterior de `data` como estructura `btree`. Sin embargo, la definición de la estructura es necesaria antes de que el puntero pueda ser referenciado para acceder a sus miembros individualmente.

`warning: unknown escape sequence '...'`

Este error es causado por un incorrecto uso del carácter de escape en una cadena. Las secuencias de escape válidas son:

`\n` nueva línea	`\t` tabulador
`\b` retroceso	`\r` retorno de carro
`\f` alimentación de línea	`\v` tabulador vertical

`\a` alerta (campana)

Las combinaciones `\\`, `\'`, `\"` y `\?` pueden ser usadas para caracteres individuales. Las secuencias de escape pueden usar códigos octales `\0`–`\377` y códigos hexadecimales `\0x00`–`\0xFF`.

Ejemplo:

```
#include <stdio.h>

int
main (void)
{
  printf ("!HOLA MUNDO!\N");
  return 0;
}
```

La secuencia de escape `\N` en el programa anterior es inválida —La correcta secuencia de escape para el carácter de nueva línea es `\n`.

`warning: suggest parentheses around assignment used as truth value`

Este aviso resalta un potencial serio error, usando el operador '`=`' en vez del operador de comparación '`==`' en el test de la instrucción condicional o en otra expresión lógica. Mientras el operador de asignación puede ser usado como parte de un valor lógico, éste es raramente el comportamiento deseado.

Ejemplo:

```
#include <stdio.h>

int
main (void)
{
  int i = 0;
  if (i = 1) {  /* = deberia ser == */
    printf ("resultado inesperado\n");
  }
  return 0;
}
```

El test anterior debería haberse escrito como `if (i == 1)`, en otro caso la variable `i` será puesta a `1` por la evaluación de la misma instrucción if. El operador '`=`' tanto asigna como devuelve el valor de su parte derecha,

causando que la variable `i` sea modificada y se tome una bifurcación inesperada. Similar resultado inesperado sucede con `if (i = 0)` en vez de `if (i == 0)`, excepto que en este caso el cuerpo de la instrucción `if` nunca será ejecutado.

Este aviso se suprime si la instrucción es encerrada entre paréntesis adicionales para indicar que está siendo usado legítimamente.

`warning: control reaches end of non-void function`

Una función que fué declarada con un tipo de retorno, tal como `int` o `double`, debe tener siempre una instrucción `return` devolviendo un valor del correspondiente tipo en todos los puntos de salida posibles —en otro caso el valor de retorno no está bien definido. Las funciones declaradas `void` no necesitan instrucciones `return`.

Ejemplo:

```
#include <stdio.h>

int
display (const char * str)
{
  printf ("%s\n", str);
}
```

El programa anterior alcanza el final de la función `display`, la cual retorna un tipo `int`, sin una instrucción `return`. Una línea adicional como `return 0;` es necesaria.

Cuando se usa `gcc` la función `main` de un programa C debe devolver un valor de tipo `int` (el estado de salida del programa). En C++ la instrucción `return` puede ser omitida en la función `main` —el valor de retorno de la función `main` por defecto es 0 si no es especificado.

`warning: unused variable '...'`
`warning: unused parameter '...'`

Este aviso indica que una variable ha sido declarada como variable local o parámetro de una función, pero no ha sido usada en ningún lugar. Una variable no usada puede ser el resultado de un error de programación, tal como usar accidentalmente el nombre de una variable en vez de otro.

Ejemplo:

```
int
foo (int k, char * p)
{
  int i, j;
  j = k;
  return j;
}
```

En este programa la variable `i` y el parámetro `p` nunca son usados. Observe que las variables no usadas son informadas por al opción '`-Wall`', mientras que los parámetros no usados sólo se muestran con '`-Wall -W`'.

`warning: passing arg of ... as ... due to prototype`
: Este aviso ocurrre cuando una función es llamada con un argumento de diferente tipo al especificado en su prototipo. La opción '`-Wconversion`' es necesaria para habilitar estos avisos. Vea la descripción de '`-Wconversion`' en Sección 3.5 [Opciones de aviso adicionales], página 37 para ver un ejemplo.

`warning: assignment of read-only location`
`warning: cast discards qualifiers from pointer target type`
`warning: assignment discards qualifiers ...`
`warning: initialization discards qualifiers ...`
`warning: return discards qualifiers ...`
: Estos avisos ocurren cuando un puntero es usado de manera incorrecta, violando un calificador de tipo tal como `const`. Los datos accedidos a través de un puntero marcado como `const` no se modificará, y el puntero en sí sólo puede ser asignado a otros punteros que también son marcados como `const`.

  Ejemplo:

```
char *
f (const char *s)
{
  *s = '\0'; /* asigna dato en modo solo */
  return s; /* lectura descartando const */
}
```

  Este programa intenta modificar datos constantes, y descartar la propiedad `const` del argumento `s` en el valor devuelto.

`initializer element is not a constant`

En C, las variables globales solo pueden ser inicializadas con constantes, tales como valores numéricos, `NULL` o cadenas. Este error ocurre si un valor no constante es usado.

Ejemplo:

```
#include <stdio.h>

FILE *stream = stdout;/* no es constante */
int i = 10;
int j = 2 * i;        /* no es constante */

int
main (void)
{
  fprintf (stream, "!Hola Mundo!\n");
  return 0;
}
```

Este programa intenta inicializar dos variables desde otras variables. En particular, no se requiere al flujo `stdout` ser una constante por el estándar de C (aunque en algunos sistemas es una constante). Nótese que inicializadore no constantes son permitidos en C++.

13.3 Mensajes de error del enlazador

`file not recognized: File format not recognized`

GCC usa la extensión de un fichero, tal como '`.c`' o '`.cc`', para determinar su contenido. Si no hay extensión GCC no puede reconocer el tipo de fichero y dará este error.

Ejemplo:

```
#include <stdio.h>

int
main (void)
{
  printf ("Hello World!\n");
  return 0;
}
```

Si el programa de debajo está guardado en un fichero '`hola`' sin extensión, entonces al compilarlo dará el siguiente error:

```
$ gcc -Wall hola
hola: file not recognized: File format not
recognized
collect2: ld returned 1 exit status
```

La solución es renombrar el fichero para la correcta extensión, en este caso '`hola.es.c`'.

`undefined reference to 'foo'`
`collect2: ld returned 1 exit status`

Este error ocurre cuando un programa usa una función o variable que no está definida en cualquiera de los ficheros objeto o librerías suministradas para el enlazador. Esto puede ser causado por una librería perdida o el uso de un nombre incorrecto. En el mensaje de error de debajo, el programa '`collect2`' es parte del enlazador.

Ejemplo:

```
int foo(void);

int
main (void)
{
  foo();
  return 0;
}
```

Si este programa está compilado sin ser enlazado a una librería o fichero objeto conteniendo la función `foo()` habrá una indefinida referencia de error.

`/usr/lib/crt1.o(.text+0x18): undefined reference to 'main'`

Este error es un caso especial del error de debajo, cuando la función perdida es `main`. En C y C++, cada programa debe tener una función `main` (dónde la ejecución comienza). Cuando se compila un fichero fuente individual sin una función `main`, usa la opción '`-c`' (véase Sección 2.4.1 [Creando archivos objeto desde archivos fuente], página 13).

13.4 Mensajes de error en tiempo de ejecución

`error while loading shared libraries:`
`cannot open shared object file: No such file or directory`
: El programa usa librerías compartidas, pero los necesarios ficheros de las librerías compartidas no pueden ser encontrados por el enlazador dinámico cuando el programa comienza. La ruta de búsqueda para las librerías compartidas es controlada por la variable de entorno `LD_LIBRARY_PATH` (véase Sección 3.2 [Librerías compartidas y librerías estáticas], página 28).

`Segmentation fault`
`Bus error`
: Estos mensajes en tiempo de ejecución indican un error de acceso a memoria

  Causas comunes incluyen:

  - desreferenciando un puntero nulo o un puntero no inicializado
  - acceso al array fuera de los límites
  - incorrecto uso de `malloc`, `free` y funciones relacionadas
  - uso de `scanf` con argumentos no válidos

  Hay una sutil diferencia entre fallos de segmentación y errores de bus. Un fallo de segmentación ocurre cuando un proceso intenta acceder a memoria protegida por el sistema operativo. Un error de bus ocurre cuando memoria válida es accedida de un modo incorrecto (por ejemplo, intentando leer un *no alineado* valor en arquitecturas dónde los valores deben ser alineados con 4-bytes).

`floating point exception`
: Este error en tiempo de ejecución es causado por una excepción aritmética, tal como división por cero, desbordamiento, por exceso y por defecto o una operación no válida (p.e. la raíz cuadrada de -1). El sistema operativo determina qué condiciones producen este error. En sistemas GNU, las funciones `feenableexcept` y `fedisableexcept` pueden ser usadas para capturar o enmascarar cada tipo de excepción.

`Illegal instruction`
: Este error es producido por el sistema operativo cuando se encuentra una instrucción máquina ilegal. Esto ocurre cuando el código ha sido compilado para una arquitectura específica y se ejecuta en otra.

14 Obteniendo ayuda

Si encuentras un problema no cubierto por esta introducción, hay varios manuales de referencia que describen GCC y asuntos relacionados con el lenguaje en más detalle (véase [Lectura adicional], página 131). Estos manuales contienen respuestas a preguntas comunes, y un cuidadoso estudio de éstas normalmente producirán una solución.

De otro modo, hay muchas compañías y consultores quienes ofrecen soporte comercial para cuestiones de programación relativas a GCC por horas o de forma contínua. En los negocios esto puede ser un camino de costes efectivo para obtener soporte de alta calidad.

Un directorio de compañías que dan soporte al software libre y sus precios actualizados puede ser encontrado en el sitio web[1] del Proyecto GNU. Con software libre, el soporte comercial está disponible en un mercado libre —compañías de servicios compiten en calidad y precio, y los usuarios no están atados a una determinada. En contraste, el soporte para el software privativo está normalmente sólo disponible desde el vendedor original.

Un soporte comercial de alto nivel para GCC está disponible desde compañías involucradas en el desarrollo del conjunto de herramientas del compilador de GNU por sí mismo. Un listado de estas compañías puede ser encontrado en la sección de “Compañías de Desarrollo” de la página web de la editorial para este libro.[2] Estas compañías pueden proporcionar servicios tales como extender GCC para generar código para nuevos procesadores o para arreglar errores encontrados en el compilador.

[1] http://www.gnu.org/prep/service.html
[2] http://www.network-theory.co.uk/gcc/intro/

Lectura adicional

La guía definitiva para GCC es el manual oficial de referencia, "*Using GCC*", publicado por GNU Press:

> *Using GCC (for GCC version 3.3.1)* por Richard M. Stallman y the GCC Developer Community (Publicado por GNU Press, ISBN 1-882114-39-6)

Este manual es esencial para cualquiera que trabaje con GCC porque describe cada opción en detalle. Nótese que el manual está actualizado cuando nuevas versiones de GCC llegan a estar disponibles, así el número ISBN puede cambiar en el futuro.

Si eres nuevo programando con GCC también querrás aprender a usar el Depurador de GNU GDB, y a cómo compilar largos programas fácilmente con GNU Make. Éstas herramientas son descritas en los siguientes manuales:

> *Debugging with GDB: The GNU Source-Level Debugger* por Richard M. Stallman, Roland Pesch, Stan Shebs, et al. (Publicado por GNU Press, ISBN 1-882114-88-4)
>
> *GNU Make: A Program for Directing Recompilation* por Richard M. Stallman y Roland McGrath (Publicado por GNU Press, ISBN 1-882114-82-5)

Para una efectiva programación en C es también esencial tener buenos conocimientos de la librería estándar de C. El siguientes manual documenta todas las funciones en la GNU C Library:

> *The GNU C Library Reference Manual* por Sandra Loosemore con Richard M. Stallman, et al (2 vols) (Publicado por GNU Press, ISBN 1-882114-22-1 y 1-882114-24-8)

Asegúrate de visitar el sitio web `http://www.gnupress.org/` para las últimas ediciones impresas de manuales publicados por GNU Press. Los manuales pueden ser adquiridos online usando una tarjeta de crédito en el sitio web de la FSF[1] además de estar disponbles para su compra a través de la mayoría de las librerías usando ISBN. Manuales publicados por GNU Press financian la Free Software Foundation y el Proyecto GNU.

Información acerca de comandos de shell, variables de entorno y reglas de comillas en shell puede ser encontrada en el siguiente libro:

> *The GNU Bash Reference Manual* por Chet Ramey y Brian Fox (Publicados por Network Theory Ltd, ISBN 0-9541617-7-7)

[1] `http://order.fsf.org/`

Otros Manuales de GNU mencionados en este libro (tal como *GNU gprof —The GNU Profiler* y *The GNU Binutils Manual* no estaban disponibles en forma impresa en el momento en que este libro fué a imprenta. Enlaces a copias online pueden ser encontradas en la página web de la editorial para esto book.[2]

La página web oficial del Proyecto GNU para GCC puede ser encontrado en `http://www.gnu.org/software/gcc/`. Éste sitio incluye una lista de preguntas frecuentes, así como la base de datos de errores y gran cantidad de otra información útil acerca de GCC.

Hay muchos libros acerca de los lenguajes C y C++. Dos de los estándares de referencia son:

> *The C Programming Language* (ANSI edition) Brian W. Kernighan, Dennis Ritchie (ISBN 0-13110362-8)
>
> *The C++ Programming Language* (3rd edition) Bjarne Stroustrup (ISBN 0-20188954-4)

Cualquiera usando los lenguajes C y C++ en un contexto profesional obtendrá una copia de los estándares oficiales, que están disponibles como libros impresos:

> *The C Standard: Incorporating Technical Corrigendum 1* (Publicado por Wiley, ISBN 0-470-84573-2)
>
> *The C++ Standard* (Publicado por Wiley, ISBN 0-470-84674-7)

Por referencia, el número estándar C es ISO/IEC 9899:1990, para el original estándar C publicado en 1990 e implementado por GCC. Un estándar de C revisado ISO/IEC 9899:1999 (conocido como C99) fué publicado en 1999, y éste es mayoritariamente (pero todavía no completamente) soportado por GCC. El estándar de C++ es ISO/IEC 14882.

El estándar de aritmética en coma flotante IEEE-754 es importante para cualquier programa involucrados en computación numérica. El estándar está disponible de manera comercial desde el IEEE, y también es descrito en el siguiente libro:

> *Numerical Computing with IEEE Floating Point Arithmetic* por Michael Overton (Publicado por SIAM, ISBN 0-89871-482-6).

El libro incluye muchos ejemplos para ilustrar la razón fundamental para el estándar.

[2] `http://www.network-theory.co.uk/gcc/intro/`

Reconocimientos

Muchas personas han contribuido a este libro, y es importante recordar sus nombres aquí:

Gracias a Gerald Pfeifer, por su cuidadosa revisión y numerosas sugerencias para mejorar el libro.

Gracias a Andreas Jaeger, por la información acerca de AMD64 y soporte multi-arquitectura, y sus muchos útiles comentarios.

Gracias a David Edelsohn, por la información acerca de la serie de procesadores POWER/PowerPC.

Gracias a Jamie Lokier, por investigar.

Gracias a Martin Leisner, Mario Pernici, Stephen Compall y Nigel Lowry, por sus útiles correcciones.

Gracias a Gerard Jungman, por sus útiles comentarios.

Gracias a Steven Rubin, por generar la imagen del chip para la cubierta con Electric.

Y de manera más importante, gracias a Richard Stallman, fundador del Proyecto GNU, por escribir GCC y hacer que sea software libre.

Organizaciones de software libre

El GCC, GNU Compiller Collection (Colección de Compiladores de GNU) es parte del Proyecto GNU, lanzado en 1984 para desarrollar un sistema operativo Unix completo que es software libre: el sistema GNU.

La Free Software Foundation (FSF) es una organización de caridad exenta de impuestos que recoge fondo para el continuo trabajo del Proyecto GNU. Está dedicada a promover el derecho a usar, estudiar, copiar, modificar, y redistribuir programas de ordenador. Uno de los mejores caminos para ayudar al desarrollo del software libre es ser un miembro asociado a la Free Software Foundation, y pagar regularmente cuotas para dar soporte a sus esfuerzos.

Los miembros asociados a la Free Software Foundation reciben muchos beneficios incluyendo noticias, admisión a los encuentros anuales de la FSF, y descuentos en libros y CDROMs publicados por GNU Press. Los costes por ser miembro son deducibles de impuestos en Estados Unidos. Para más información acerca de cómo llegar a ser miembro visita el sitio web de la FSF en `http://www.fsf.org/`.

La Free Software Foundation Europe (FSFE) es una organización hermana de la Free Software Foundation. La FSFE es activa en la promoción del software libre a todos los niveles en Europa. Por una cuota anual, se puede ser miembro de la FSFE y dar soporte a su trabajo. Los miembros reciben una tarjeta de miembro personalizada y compatible con GPG, permitiendo autenticación digital segura de correo electrónico y ficheros, y obtener acceso a la "FSFE Fellowship", una comunidad electrónica por la libertad del software. Para más información, visita el sitio web de la FSFE en `http://www.fsfe.org/`.

La *Foundation for a Free Information Infrastructure (FFII)* es otra organización importante en Europa. FFII no se dedica específicamente al software libre, pero trabaja para defender los derechos de todos los programadores y usuarios de ordenadores contra los monopolios en el campo de la computación, tales como patentes de software. Para más información acerca de FFII, o para dar soporte a su trabajo con una donación, visita su sitio web en `http://www.ffii.org/`.

Licencia para documentación libre GNU

Version 1.2, November 2002

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

   B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

   C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

   D. Preserve all the copyright notices of the Document.

   E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

   You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

   The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

   In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a

disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C) year your name.
Permission is granted to copy, distribute and/or modify
this document under the terms of the GNU Free
Documentation License, Version 1.2 or any later version
published by the Free Software Foundation; with no
Invariant Sections, no Front-Cover Texts, and no
Back-Cover Texts.  A copy of the license is included in
the section entitled ``GNU Free Documentation License''.
```

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

```
with the Invariant Sections being list their
titles, with the Front-Cover Texts being list, and
with the Back-Cover Texts being list.
```

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Índice

#

#define, directiva del preprocesador 43
#elif, directiva del precompilador 116
#else, directiva del preprocesador 116
#if, directiva del preprocesador .. 35
#ifdef, directiva del preprocesador 43
#include, directiva del preprocesador 12

$

$, prompt de shell 7

-

'**--help**', opción para mostrar las opciones de línea de comandos 91
--version, opción para mostrar el número de versión 91
'**-ansi**', opción que inhabilita extensiones del lenguaje 31
'**-ansi**', opción usada con **g++** 69
'**-c**', opción para compilar a fichero objeto 13
'**-D**', opción para definir macros... 43
'**-dM**', opción que lista las macros predefinidas 44
'**-E**', opción para preprocesar ficheros fuente 46
'**-fno-default-inline**', opción... 71
'**-fno-implicit-templates**', opción para deshabilitar la instanciación de plantillas 76
'**-fprofile-arcs**', opción para instrucciones de bifurcación 104
'**-fsigned-bitfields**', opción 90
'**-fsigned-char**', opción 88
'**-ftest-coverage**', opción para registro de cobertura 104
'**-funroll-loops**', opción para hacer optimización por desenrollado de bucles 63
'**-funsigned-bitfields**', opción .. 90
'**-funsigned-char**', opción 88
'**-g**', opción que activa la depuración 49
'**-I**', opción para asignar la ruta de include 24
'**-L**', opción de ruta de búsqueda de librería 24
'**-l**', opción para enlazar con librerías 19
'**-lm**', opción de enlace con librería matemática 19
'**-m**', opción de configuración específicas de la plataforma .. 79
'**-m32**' y '**-m64**', opciones para compilar en entornos de 32 o 64 bits..... 83
'**-maltivec**', opción que habilita el uso del procesador Altivec en PowerPC 83
'**-march**', opción para compilación para una CPU específica 79
'**-mcmodel**', opción para AMD64 .. 81
'**-mcpu**', opción para compilar en una CPU específica 83
'**-mfpmath**', opción para aritmética en coma flotante 80
'**-mieee**', opción para soporte de coma flotante en DEC Alpha 81
'**-mminimal-toc**', opción en AIX .. 83
'**-mno-fused-madd**', opción en PowerPC 83
'**-msse**' y opciones relacionadas ... 80
'**-mtune**', opción 79
'**-mxl-call**', opción para compatibilidad con compiladores IBM XL en AIX 83
'**-o**', opción para asignar el nombre del fichero de salida 9
'**-O0**', opción para establece nivel de optimización cero 62

‘-O1’, opción para establecer el nivel de optimización uno 62
‘-O2’, opción para establecer el nivel de optimización dos 62
‘-O3’, opción para establecer el nivel de optimización tres 63
‘-Os’, opción para hacer optimización por tamaño 63
‘-pedantic’, opción conforme al estándar ANSI (con ‘-ansi’) 31
‘-pg’, opción para habilitar profiling 103
‘-pthread’, opción en AIX 83
‘-rpath’, opción para asignar ruta de búsqueda de librería compartida en tiempo de ejecución 29
‘-S’, opción crear código ensamblador 108
‘-save-temps’, opción que guarda ficheros intermedios 48
‘-static’, opción que fuerza el enlace estático 30
‘-std’, opción que selecciona un estándar específico del lenguaje 31, 34
‘-v’, opción para compilación verbosa 91
‘-W’, opción que habilita avisos adicionales 37
‘-Wall’, opción que habilita los avisos comunes 9
‘-Wcast-qual’, opción de aviso de casts eliminando calificadores 40
‘-Wcomment’, opción de aviso acerca de comentarios anidados 35
‘-Wconversion’, opción de aviso de conversiones de tipos 38
‘-Weffc++’, opción 71
‘-Werror’, opción que convierte avisos en errores 41
‘-Wformat’, opción de aviso de formato de cadenas incorrecto 35
‘-Wimplicit’, opción de aviso de declaraciones no encontradas 35
‘-Wold-style-cast’ option 71
‘-Wreturn-type’, opción de aviso de tipos devueltos incorrectos ... 36
‘-Wshadow’, opción de aviso de variables ocultas 39
‘-Wtraditional’, opción de aviso de C tradicional 40
‘-Wuninitialized’, opción de aviso de variables no inicializadas .. 66
‘-Wunused’, opción de aviso de variable no usada 35
‘-Wwrite-strings’, opción de aviso para cadenas constantes modificadas 40

•

.a, extensión de fichero de archivo 18
.c, extensión de fichero fuente C ... 9
.cc, extensión de fichero C++ 69
.cpp, extensión de fichero C++ 69
.cxx, extensión de fichero C++ 69
.h, extensión de fichero de cabecera 12
.i, extensión de archivo preprocesado para C 108
.ii, extensión de archivo preprocesado para C++ 108
.o, extensión de fichero objeto 13
.s, extensión de archivo ensamblador 108
.so, extensión de fichero objeto compartido 28

_

__gxx_personality_v0, error de referencia indefinida 70

A

a, extensión de fichero de archivo 18
a.out, nombre del fichero ejecutable por defecto 9
ADA, compilador gnat 5
AIX, compatibilidad con compiladores IBM XL 83

AIX, error de desboramiento TOC 83
AIX, opciones específicas de la plataforma 83
Alpha, opciones específicas de la plataforma 81
Altivec, en PowerPC 83
AMD x86, opciones específicas de la plataforma 79
AMD64, opciones específicas para procesador de 64 bits 81
análisis de flujo de datos 66
'ansi', opción que inhabilita extensiones del lenguaje 31
'ansi', opción usada con g++ 69
ANSI/ISO C estricto, opción '-pedantic' 33
ANSI/ISO C, comparado con extensiones GNU C 31
ANSI/ISO C, controlado con la opción '-ansi' 31
ANSI/ISO C, opción de diagnósticos pedantic 33
ar, GNU archiver 18, 99
Archivador de GNU, ar 18
archivador, ar 107
archivo de cabecera, con guardas include 75
archivo de cabecera, sin extensión .h para C++ 72
archivo objeto, examinando con comando file 111
archivos compilados, examinando 111
archivos de configuración de GCC 92
argumento de diferente tipo, aviso de formato 10
aritmética en coma flotante 87
aritmética en coma flotante, con extensiones SSE 80
aritmética IEEE 84
aritmética, coma flotante 84
ARM, soporte a múltiples arquitecturas 84
arrays de tamaño variable 33
asm, palabra reservada de extensión 32, 85
assignment discards qualifiers ... 124
assignment of read-only location 124
Athlon, opciones específias de la plataforma 79
attach, depuración de un programa en ejecución 95
aviso de prototipos perdidos 35
aviso de variable no usada ... 35, 123
aviso, formato de diferente tipo que el argumento 10
avisos, addicionales con '-W' 37
avisos, promoviendo errores 41
avisos, y optimización 66

B

backtrace, comando para depurar 52
benchmarking, con comando time 64
big-endian, ordenación de palabras 111
Binutils, GNU Binary Tools 112
bits, 32 versus 64 en UltraSPARC 83
break, comando en gdb 53
bucle infinito, parando 95
buffer circular, plantilla de ejemplo 74
buffer, plantilla de ejemplo 74
bus error 127

C

C library, standard 18, 132
C math library 18
C standard library 18
C tradicional (K&R), avisos de comportamiento diferente 40
C y C++ estándares en forma impresa 132
C++, compilador g++ 5
C++, compilando un pequeõ programa con g++ 69
C++, creando librerías con instanciación explícita 77
C++, espacio de nombres std 72
C++, extensiones de fichero 69

C++, g++ ejemplo de un compilador verdadero ... 69
C++, instanciación de plantillas ... 74
C++, librería estándar ... 72, 73
C++, librería estándar de plantillas ... 73
C++, plantillas ... 72
C, compilador gcc ... 5
C, compilando con gcc ... 9
c, extensión de fichero fuente C ... 9
'c', opción para compilar a fichero objeto ... 13
C/C++, riesgos de uso ... 7, 11, 21, 67
C_INCLUDE_PATH ... 26
c89/c99, seleccionado con '-std' ... 34
cadenas constantes con posibilidad de escritura, deshabilitando ... 40
cadenas constantes, avisos en tiempo de compilación ... 40
calificadores, aviso de sobrescritura por casts ... 40
campos orientados a bit, portabilidad con signo versus sin signo ... 90
cannot find *library* error ... 23, 25
cannot open shared object file ... 28, 127
carga dinámica ... 28
cast discards qualifiers from pointer target type ... 124
casts, usados para evitar avisos de conversión ... 39
cc, extensión de fichero C++ ... 69
CC, variable make ... 16
cero, desde el desbordamienbo inferior en DEC Alpha ... 81
cero, división por ... 82
CFLAGS, variable make ... 16
char, portabilidad con signo versus sin signo ... 87
character constant too long ... 119, 120
código exit, mostrado en gdb ... 54
código fuente ... 9
código máquina ... 9
código no optimizado ('-O0') ... 62
collect2: ld returned 1 exit status ... 126
coma flotante, problemas de portabilidad ... 84
comando file, para identificación de archivos ... 111
comando nm ... 112
comando strip ... 112
comando time, midiendo tiempo de ejecución ... 64
comando ulimit ... 51, 97
comando, en makefile ... 16
comentarios anidados, aviso de ... 35
comentarios, anidados ... 35
comercial, soporte ... 129
comillas en la shell ... 46, 131
comillas, para definir una macro vacía ... 46
'comment', opción de aviso de comentarios anidados ... 35
comparación de expresiones de aviso always true/false ... 37
compila a fichero objeto, opción '-c' ... 13
compilación verbosa, opción '-v' ... 92
compilación, etapas internas de ... 107
compilación, modelo para plantillas ... 74
compilación, opciones ... 23
compilación, para depuración ... 49
compilación, parando en avisos ... 41
compilador, como funciona internamente ... 107
compilador, mensajes de error ... 116
Compiladores de GNU, principales funcionalidades ... 6
Compiladores GNU, Manual de Referencia ... 131
compiladores IBM XL, compatibilidad en AIX ... 83
compiladores XL, compatibilidad en AIX ... 83
compilando con optimización ... 57
compilando ficheros de manera independiente ... 13
compilando múltiples ficheros ... 11
compilando programas C con gcc ... 9
compilando programas C++ con g++ ... 69
compilar, convertir código fuente a código ensamblador ... 108
const, aviso de sobreescritura por casts ... 40

constante carácter multi-carácter 120
`continue`, comando en `gdb` 54
control reaches end of non-void function 123
control-C, interrumpir 95
convenciones, usadas en manual ... 7
conversiones entre tipos, aviso de 38
coversiones de tipo, aviso de de ... 38
`CPLUS_INCLUDE_PATH` 26
`cpp`, extensión de fichero C++ 69
`cpp`, preprocesador de C 43
`CPPFLAGS`, variable make 16
'`cr`', opción para crear/reemplazar archivos de librerías 100
creando ficheros ejecutables desde ficheros objeto 14
creando ficheros objeto desde ficheros fuente 13
`cxx`, extensión de fichero C++ 69
`CXX`, variable make 16
`CXXFLAGS`, variable make 16

D

'`D`', opción para definir macros 43
DEC Alpha, opciones específicas de la plataforma 81
declaración implícita de función .. 22, 35, 118
declaración, en fichero de cabecera 12
declaración, perdida 21
definiendo macros 43
dependencia, en makefile 16
dependencias, de librerías compartidas 113
depuración, con `gdb` 49
depurando, con optimización 66
depurando, flags de compilación .. 49
dereferencing pointer to incomplete type 121
desbordamiento inferior, aritmética de coma flotante 85
desbordamiento por defecto gradual, en DEC Alpha 81
desbordamiento por defecto suave, en DEC Alpha 81
desbordamiento por defecto, en DEC Alpha 81
desbordamiento, aritmética de coma flotante 85
Desenrollado de bucle, optimización 60, 63
desenrollado, bucles (optimización) 60, 63
despliegue, opciones para .. 49, 62, 66
desreferenciando, puntero nulo 50
dialectos del lenguaje C 31
diferencias numéricas 84
dilema velocidad-espacio, en optimización 60
dilema, entre velocidad y espacio en optimización 60
directiva de preprocesamiento inválida 116
directorio '`/tmp`', ficheros temporales 19
directorios por defecto, enlazando y ficheros de cabecera 23
división por cero 82
DLL (dynamically linked library), ver librerías compartidas 28
'`dM`', opción que lista de macros predefinidas 44
doble precisión 84

E

'`E`', opción para preprocesar ficheros fuente 46
EGCS (Experimental GNU Compiler Suite) 5
ejecutable instrumentado, para análisis 103
ejecutable instrumentado, para test de cobertura 104
ejecutable, creando ficheros objeto al enlazar 14
ejecutable, funcionando 10
ejecutable, nombre de fichero por defecto `a.out` 9
ejecutable, tabla de símbolos almacenada en 49
ejecutable. examinando con comando `file` 111
ejecutando un fichero binario, C .. 10

ejecutando un fichero ejecutable, C++ 69
ejemplos, convenciones usadas 7
eliminación de subexpresión común, optimización 57
eliminación de subexpresión, optimización 57
eliminación, de subexpresiones comunes 57
Emacs, modo `gdb` 55
endianness, ordenación de palabras 111
enlace, explicación de 13
enlazado, error de referencia indefinida debido al orden de enlace de librerías 20
enlazador, descripción inicial 14
enlazador, GNU comparado con otros enlazadores 76
enlazador, `ld` 107, 109
enlazador, mensajes de error 125
enlazando estáticamente, se fuerza con '`-static`' 30
enlazando, con librería usando '`-l`' 19
enlazando, con librerías externas .. 18
enlazando, creando ficheros ejecutables desde ficheros objeto 14
enlazando, dinámicamente (librerías compartidas) 28
enlazando, directorios por defecto 23
enlazando, ficheros objeto actualizados 15
ensamblador, `as` 107
ensamblador, convirtiendo lenguaje ensamblador en código máquina 109
entero con signo, casting 39
entero sin signo, casting 39
error de análisis sintáctico debido a extensiones del lenguaje 32
error de definición múltiple de símbolo, con C++ 76
error de desbordamiento TOC, en AIX 83
error de desbordamiento, para TOC en AIX 83
error de enlace, no se puede encontrar librería 23
error de identificador no declarado para la librería C, al usar la opción '`-ansi`' 32
error de instrucción ilegal 80, 128
error de referencia indefinida para `__gxx_personality_v0` 70
error de referencia no definida ... 19, 20, 126
error while loading shared libraries 28, 127
error, ejemplo de 11, 21, 50
error, referencia indefinida debido al orden de enlace de librerías .. 20
errores comunes, no incluidos con '`-Wall`' 37
espacio de nombres `std` en C++ ... 72
espacio de nombres, prefijo reservado para el preprocesador 44
espacio en disco, uso reducido por librerías compartidas 28
espacio versus velocidad, dilema en optimización 60
estándar IEEE-754 132
estándares C, C++ y aritmética del IEEE 132
estándares del lenguaje, seleccionando con '`-std`' 34
etapas de compilaci'on, usadas internamente 107
examinando archivos compilados 111
examinando ficheros core 49
excepción de coma flotante, en Alpha DEC 82
expansión de función en línea, ejemplo de optimización 58
expansión en línea, ejemplo de optimización 58
extensión de archivo, archivo ensamblador `.s` 108
extensión de archivo, archivo preprocesado `.i` 108
extensión de archivo, archivo preprocesado `.ii` 108
extensión de fichero, `.h` fichero de cabecera 12

extensión de fichero, fichero fuente `.c` 9
extensión de fichero, fichero objeto `.o` 13
extensión de fichero, fichero objeto compartido `.so` 28
extensión del fichero, fichero de archivo `.a` 18
extensión, `.C`, fichero C++ 69
extensión, `.cc`, fichero C++ 69
extensión, `.cpp`, fichero C++ 69
extensión, `.cxx`, fichero C++ 69
extensión, archivo ensamblador `.s` 108
extensión, archivo preprocesado `.i` 108
extensión, archivo preprocesado `.ii` 108
extensión, fichero de archivo `.a` ... 18
extensión, fichero de cabecera `.h` .. 12
extensión, fichero fuente `.c` 9
extensión, fichero objeto `.o` 13
extensión, fichero objeto compartido `.so` 28
extensiones BSD, GNU C Library 33
extensiones GNU C, comparadas con ANSI/ISO C 31
extensiones MMX 80
extensiones POSIX, GNU C Library 33
extensiones SSE 80
extensiones SVID, GNU C Library 33
extensiones XOPEN, GNU C Library 33

F

fallos de programa, salvado en archivo core 49
fallos, salvado en archivo core 49
fichero `bash` profile 51
fichero `bash` profile, configuraciones de login 26, 30
fichero binario, también llamado fichero ejecutable 9
fichero core, examinando 49, 51
fichero core, no se produce 51
fichero DBM, creado con `gdbm` 24
fichero de archivo, explicación de 18
fichero de archivo, extensión `.a` ... 18
fichero de cabecera perdida, causa declaración implícita 22
fichero de cabecera, declaraciones 12
fichero de cabecera, directorios por defecto 23
fichero de cabecera, extensión `.h` .. 12
fichero de cabecera, la cabecera perdida causa declaración implícita 22
fichero de cabecera, no compilado 14
fichero de cabecera, no encontrado 23
fichero de cabecera, perdido 21
fichero de cabecera, ruta de include —extendiendo con '`-I`' 24
fichero de configuración del cargador, `ld.so.conf` 30
fichero de login, asignando variables de entorno en él 29
fichero ejecutable 9
fichero fuente C, extensión `.c` 9
fichero objeto compartido, extensión `.so` 28
fichero objeto, creando desde las fuentes usando la opción '`-c`' 13
fichero objeto, explicación de 13
fichero objeto, extensión `.o` 13
fichero profile, asignando variables de entorno en él 29
ficheros de cabecera perdidos 21
ficheros fuente actualizados, recompilando 15
ficheros fuente modificados, recompilando 15
ficheros fuente, recompilando 15
ficheros intermedios, guardando .. 48
ficheros objeto actualizados, reenlazando 15
ficheros objeto, enlazando para crear un fichero ejecutable 14
ficheros objeto, reenlazando 15
ficheros objeto, temporal 19

ficheros preprocesados, guardando 48
ficheros temporales, escritos en '/tmp' 19
ficheros temporales, guardando ... 48
file extension, .C, fichero C++ 69
file extension, .cc, fichero C++ 69
file extension, .cpp, fichero C++ .. 69
file extension, .cxx, fichero C++ .. 69
file format not recognized 125
file not recognized 125
finish, comando en gdb 54
fldcw activar modo de coma flotante 85
floating point exception 127
'fno-default-inline', opción 71
'fno-implicit-templates', opción para deshabilitar la instanciación de plantillas 76
forma impresa del estándar de aritmética del IEEE 132
formato COFF 111
formato de cadenas, aviso de uso incorrecto 35
formato ELF 111
formato, aviso de tipo diferente en el argumento 10
Fortran, compilador g77 5
'fpmath', opción para aritmética en coma flotante 80
'fprofile-arcs', opción para instrucciones de salto 104
Free Software Foundation (FSF) ... 5
FreeBSD, aritmética de coma flotante 85
'ftest-coverage', opción para registro de cobertura 104
función de carga 28
funcionalidades, de GCC 6
'funroll-loops', opción para hacer optimización por desenrollado del bucles 63

G

g++, compilando programas C++ .. 69
g++, GNU C++ Compiler 5
'g', opción que activa la depuración 49
g77, compilador de Fortran 5
gcc, ejemplo simple 9
gcc, GNU C Compiler 5
gcc, usado inconsistentemente con g++ 70
gcj, GNU Compiler for Java 5
gcov, GNU coverage testing tool 104
gdb 49
gdb, depurando ficheros core con .. 51
gdb, interfaz gráfica 55
gdb, modo Emacs 55
gdbm, GNU DBM library 24
gmon.out, archivo de datos para gprof 103
gnat, compilador GNU de ADA ... 5
GNU C Library, macros de test de funcionalidad 33
GNU debugger, gdb 49
GNU Make 16
GNU/Linux, aritmética de coma flotante 85
gnu89/gnu99, seleccionado con '-std' 34
gprof, GNU Profiler 101
guardas include, en archivo de cabecera 75
gxx_personality_v0, error de referencia indefinida 70

H

h, extensión de fichero de cabecera 12
habilitar profiling, opción '-pg' .. 103
hilos, en AIX 83
historia, de GCC 5
Hola Mundo, programa en C 9
Hola Mundo, programa en C++ ... 69

I

i, extensión de archivo preprocesado para C 108
'I', opción para asignar la ruta de include 24
identificación de archivos, con comando file 111

identificador de proceso, encontrando 95
ii, extensión de archivo preprocesado para C++ 108
include nested too deeply 115
independiente compilación de ficheros 13
Inf, infinito, en DEC Alpha 81
initialization discards qualifiers .. 124
initialization makes integer from pointer without a cast 120
initializer element is not a constant 125
Insight, Interfaz gráfica para gdb .. 55
instanciación explícita de plantillas 76
instanciación, de plantillas en C++ 74
instanciación, explícita versus implícita en C++ 76
instrucción combinada de multiplicación y suma 83
instrucción fusionada de multiplicación y suma 83
instrucción máquina, palabra reservada asm 85
Intel x86, opciones específicas de la plataforma 79
ISO C++, controlado con opción '-ansi' 69
ISO C, comparado con extensiones GNU C 31
ISO C, controlado con la opción '-ansi' 31
iso9899:1990/iso9899:1999, seleccionado con '-std' 34
Itanium, soporte a múltiples plataformas 84

J

Java, comparado con C/C++ 7
Java, compilador gcj 5

K

K&R dialecto de C, avisos de comportamiento diferente 40
kernel Linux, coma flotante 85
Kernighan and Ritchie, *The C Programming Language* 132

L

'L', opción de ruta de búsqueda de librerí 24
'l', opción para enlazar con librerías 19
ld returned 1 exit status 126
ld.so.conf, fichero de configuración del cargador 30
ld: cannot find library error 23
LD_LIBRARY_PATH, ruta de carga de librería compartida 29
ldd, cargador dinámico 113
lenguaje C, dialectos de 31
libreías compartidas, dependencias 113
librería de ficheros de cabecera, usando 21
librería dinámicamente enlazada, ver librerías compartidas 28
librería estándar, C++ 72
librería matemática 18
librería matemática, enlazando con '-lm' 19
librería, C math library 18
librería, C standard library 18
librería, libraría estándar C++ 72
librerías compartidas 28
librerías compartidas, asignando ruta de carga 29
librerías compartidas, error al cargarse 28
librerías compartidas, examinado con ldd 113
librerías compartidas, ventajas de 28
librerías de sistema 18
librerías de sistemas, localización de 18, 23, 84
librerías enlazadas dinámicamente, examinando con ldd 113
librerías estáticas 28
librerías externas, enlazando con .. 18
librerías, almacenado en ficheros de archivo 18
librerías, creando con ar 99

librerías, creando con instanciación explícita en C++ 77
librerías, en plataformas de 64 bits 23
librerías, encontrando dependencias de librerías compartidas 113
librerías, enlazando con 18, 19
librerías, error al cargar librería compartida 28
librerías, error de enlace debido a referencia indefinida 19
librerías, extendiendo ruta de búsqueda con '-L' 24
librerías, orden de enlace 20
libros de referencia 132
libros, lectura adicional 132
libstdc++, librería estándar C++ 73
Lisp, comparado con C/C++ 7
little-endian, ordenación de palabras 111
los estándares ANSI para los lenguajes C y C++, disponibles como libros 132
los estándares ISO para los lenguajes C y C++, disponibles como libros 132
LSB, byte menos significativo (least significant byte) 111

M

'm', opción de configuración específica de la plataforma 79
'm', opción para compilación para una CPU específica 79
'm32' y 'm64', opciones para compilar en entornos de 32 o 64 bits... 83
macro _GNU_SOURCE, habilita extensiones a la GNU C Library 33
macro GNU_SOURCE (_GNU_SOURCE), habilita extensiones a la GNU C Library 33
macro indefinida, comparada con macro vacía 46
macro or '#include' recursion too deep 115
macro vacía, comparada con macro indefinida 46
macros de test de funcionalidad, GNU C Library 33
macros del preprocesador, valor por defecto de 46
macros predefinidas 44
macros predefinidas específicas del sistema 44
macros, definición en preprocesador 43
macros, definidas con valor 44
macros, valor por defecto de 46
makefile, ejemplo de 16
'maltivec', opción que habilita el uso del procesador Altivec en PowerPC 83
manejando de excepción de coma flotante 86
manejando excepción, coma flotante 86
Manual de GDB 131
Manual de GNU Make 131
Manual de Referencia de la GNU C Library 131
manuales GNU Press 131
manuales para software GNU 131
más lectura acerca del lenguaje C 132
'mcmodel', opción para AMD64 ... 81
'mcpu', opción para compilar en una CPU específica 83
mejoras de GCC 129
mensajes comunes de error 115
Mensajes de error en tiempo de ejecución 127
mensajes de error, ejemplos comunes 115
'mfpmath', opción para aritmética en coma flotante 80
'mieee', opción para soporte de coma flotante en DEC Alpha 81
MIPS64, soporte a múltiples arquitecturas 84
'mminimal-toc', opción en AIX ... 83
'mno-fused-madd', opción en PowerPC 83
modelo de compilación de inclusión, en C++ 74

modo kernel, en AMD64 81
mostrando una traza 52
Motorola 680x0, aritmética en coma flotante 84
Motorola 680x0, ordenación de palabras 111
MSB, byte más significativo (most significant byte) 111
'`msse`' y opciones relacionadas 80
'`mtune`', opción 79
múltiples directorios, al incluir y rutas de enlace 27
múltiples ficheros, compilando 11
multiplicar y añadir instrucción .. 83
'`mxl-call`', opción para compatibilidad con compiladores IBM XL en AIX 83

N

NaN, no es un número, en DEC Alpha 81
NetBSD, aritmética de coma flotante 85
`next`, comando en `gdb` 53
nivel de parche, de GCC 91
niveles de optimización 62
No such file or directory 115, 127
No such file or directory, header file not found 23, 25
'`no-default-inline`', opción 71
nombre del fichero ejecutable por defecto, `a.out` 9
número de versión de GCC, mostrando 91
número de versión mayor, de GCC 91
número de versión menor, de GCC 91
números de línea, grabados en ficheros preprocesados 47
números no normalizados, en DEC Alpha 81

O

`o`, extensión de fichero objeto 13
'`o`', opción para asignar el nombre del fichero de salida 9
'`O`', opción para establecer el nivel de optimización 62
Objective-C 5
objetivo, en makefile 16
obteniendo ayuda 129
ocultamiento de variables 39
opción `char` con signo 87
opción `char` sin signo 87
opción de aviso '`effc++`' 71
opción de aviso '`old-style-cast`' 71
opción de aviso, avisos adicionales '`-W`' 37
opción de ayuda en línea de comandos 91
opción de ayuda verbosa 91
opción para asignar fichero de salida, '`-o`' 9
opción para campos orientados a bit con signo 90
opción para campos orientados a bit sin signo 90
opciones de aviso, '`-Wall`' 9
opciones de aviso, en detalle 35
opciones de avisos adicionales 37
opciones de avisos, adicionales 37
opciones de ayuda 91
opciones de resolución de problemas 91
opciones específicas de la máquina 79
opciones específicas de la plataforma 79
opciones específicas para procesador de 64 bits, AMD e Intel 81
opciones IEEE, en DEC Alpha ... 81
opciones, compilación 23
opciones, específicas de la plataforma 79
OpenBSD; aritmética de coma flotante 85
optimización a nivel de código 57
optimización por tamaño, opción '`-Os`' 63
optimización, compilando con '`-O`' 62
optimización, desenrollado de bucle 60, 63

optimización, dilema velocidad-espacio 60
optimización, ejemplo de 63
optimización, eliminación de subexpresión común 57
optimización, explicación de 57
optimización, niveles de 62
optimización, y avisos del compilador 66
optmización, con depuración 66
orden de enlace, de izquierda a derecha 20
orden de enlace, de librerías 20
ordenación de palabras, endianness 111
ordenando librerías 20

P

palabra reservada `export`, no soportada en GCC 77
palabras reservadas, adicionales en GNU C 32
parando la ejecución, con puntos de ruptura en `gdb` 53
pares clave valor, almacenados con GDBM 24
parse error 117
parse error at end of input 117
passing arg of function as another type to prototype 124
'`pedantic`', opción 31, 33
Pentium, opciones específicas de la plataforma 79
'`pg`', opción para habilitar profiling 103
pila de rastreo, mostrando 52
planificación de instrucción, optimización 61
planificación, fase de optimización 61
plantillas, en C++ 72
plantillas, instanciación esplícita .. 76
plantillas, modelo de compilación de inclusión 74
plantillas, palabra reservada `export` 77
plataformas de 64 bits, directorios de librería adicionales 23
PowerPC y POWER, opciones específicas de la plataforma .. 83
PowerPC, soporte a múltiples arquitecturas 84
precedencia, al usar preprocesador 45
precisión extendida, procesadores x86 84
precisión SSE/SSE2 87
predefinidas, macros 44
preprocesador, `cpp` 107
Preprocesador, mensajes de error 115
preprocesador, primera etapa de compilación 108
preprocesador, usando 43
preprocesamiento de ficheros fuente, opción '`-E`' 46
principales funcionalidades, de GCC 6
`print` comando para depurar 52
`printf`, aviso de uso incorrecto ... 35
`printf`, ejemplo de error por formato 10
procesadores con doble precisión nativa 84
profiling, con `gprof` 101
programa C simple, compilando ... 9
programa C++ simple, compilando 69
programación genérica, en C++ ... 72
programas C, recompilando después de modificarse 15
prompt de shell 7
prototipos, perdidos 35
Proyecto GNU, historia de 5
'`pthread`', opción en AIX 83
puntero no inicializado 127
puntero nulo 50, 127
puntos de ruptura, definición de .. 53

R

recompilando ficheros fuente modificados 15
redondeo, aritmética de coma flotante 85
reenlazando ficheros objeto actualizados 15

referencia indefinida a 'main' 126
referencia indefinida a función en C++, ocurre enlazando con `gcc` 70
referencia, no definida debido a librería perdida 19
reglas implícitas, en makefile 16
reglas, en makefile 16
return discards qualifiers 124
`return` vacío, uso incorrecto de ... 36
Richard Stallman, principal autor de GCC 5
riesgos, ejemplos de 7, 11
'`rpath`', opción para asignar ruta de búsqueda de librería compartida en tiempo de ejecución 29
ruta de enlace, asignación con variable de entorno 26
ruta de include, extendiendo con '`-I`' 24
ruta include de C 26
ruta include de C++ 26
ruta include, asignación con variables de entorno 26
rutas de búsqueda 23
rutas de búsqueda extendida, para incluir y directorios de enlace 27
rutas de búsqueda, ejemplo 24
rutas de búsqueda, extendida 27
rutas, búsqueda 23
rutas, extendiendo la variable de entorno 30

S

`s`, extensión de archivo ensamblador 108
'`S`', opción para crear código ensamblador 108
saltos, instrumentando para test de cobertura 104
'`save-temps`', opción que guarda ficheros intermedios 48
`scanf`, aviso de uso incorrecto ... 35, 127
Scheme, comparado con C/C++ 7
secuencia de Collatz 101
segmentación, explicación de 61
segmentation fault 50, 127
seleccionando estándares específicos del lenguaje, con '`-std`' 34
señal SIGINT 95
señal SIGQUIT 96
separador, en makefiles 16
`set`, comando en `gdb` 54
signo del dólar `$`, shell prompt 7
sistemas embebido, para compilación cruzada 6
Smalltalk, comparado con C/C++ .. 7
`so`, extensión de fichero objeto compartido 28
sobrecarga de las llamadas a funciones 58
sobrecarga, desde llamada a función 58
soporte a múltiples arquitecturas, discusión de 84
soporte comercial 129
SPARC, opciones específicas de la plataforma 83
Sparc64, soporte a múltiples arquitecturas 84
`specs`, directorio de archivos de configuración del compilador 92
`sqrt`, ejemplo de enlazando con .. 18
standard library, C 18
Standard Template Library (STL) 73
'`static`', opción que fuerza el enlace estático 30
`std` espacio de nombres en C++ ... 72
'`std`', opción que selecciona un estándar específico del lenguaje 31, 34
`step`, comando en `gdb` 53
suggest parentheses around assignment used as truth value 122
Sun SPARC, opciones específicas de la plataforma 83
syntax error 117
SYSV, formato de ejecutable en System V 112

T

'`t`', opción para ver la tabla de contenidos de una librería... 100
tabla de contenidos, en librerías `ar` 100
tabla de contenidos, error de desbordamiento en AIX..... 83
tabla de símbolos 49
tabla de símbolos, examinando con `nm`..... 112
tabulador, en makefiles..... 16
tamaño de palabra, en UltraSPARC 83
tamaño variable, arrays 33
tamaño, optimización por, '`-Os`' .. 63
tamaõ de palabra, determinado desde el archivo ejecutable 111
`tcsh`, comando limit..... 51
terminación, anormal (`core dumped`) 49
test de cobertura, con `gcov`..... 104
Thumb, formato alternativo de código en ARM..... 84
tiempo de ejecución, midiendo con comando `time`..... 64
tipo devuelto, no válido 36
traductores, desde C++ a C, comparados con `g++` 69
traza, mostrando..... 52
'`tune`', opción para una máquina específica..... 79
`typeof`, palabra reservada de la extensión GNU C..... 32

U

UltraSPARC, modo 32 bits versus modo 64 bits 83
`unix`, palabra reservada de la extensión GNU C..... 32
unknown escape sequence 121
unterminated string or character constant..... 118
unused parameter warning 123
Usando GCC (Manual de Referencia) 131
uso de memoria virtual, limitando 97
uso de memoria, limitando 97
utilidades relativas al compilador 99
utilidades, relativas al compilador 99

V

'`v`', opción para compilación verbosa 91
valor por defecto, de macro definido con '`-D`' 46
valor, de macro 44
variable con signo convertida a sin signo, aviso de..... 38
variable no declarada..... 116
variable no inicializada, aviso de.. 67
variable oculta 39
variable sin signo convertida a con signo, aviso de..... 38
variable, aviso de uso sin inicializar 67
variables de entorno 8, 29
variables de entorno, asignación permanente..... 29
variables de entorno, extendiendo una ruta existente 30
variables de entorno, por defecto rutas de búsqueda 26
variables de shell..... 8, 26, 29
variables shell, asignación permanente..... 29
variables, en make..... 16
`vax`, palabra reservada de la extensión GNU C..... 32
void `return`, uso incorrecto de.... 36

W

'`W`', opción que habilita avisos adicionales..... 37
'`Wall`', opción que habilita los avisos comunes..... 9
'`Wcast-qual`', opción de aviso de casts eliminando calificadores 40
'`Wconversion`', opción de aviso de conversiones de tipos 38
'`Weffc++`', opción..... 71

'`Werror`', opción que convierte avisos en errores 41
'`Wimplicit`', opción de aviso de declaraciones no encontradas 35
'`Wold-style-cast`', opción 71
'`Wreturn-type`', opción de aviso de tipos devueltos incorrectos ... 36
'`Wshadow`', opción de aviso de variables ocultas 39
'`Wtraditional`', opción de aviso de C tradicional 40
'`Wuninitialized`', opción de aviso de variables no inicializadas 66
'`Wunused`', opción de aviso de variable no usada 35
'`Wwrite-strings`', opción de aviso para cadenas constantes modificadas 40

X

x86, aritmética en coma flotante .. 84
x86, opciones específicas de la plataforma 79

Z

zona roja, en AMD64 81