Errores de compilación comunes en gcc

Paco Abad

22 de octubre de 2002

Índice

1. Introducción

El compilador `gcc` puede producir dos tipos de mensajes: errores y avisos (*warning*). Cada tipo tiene un propósito distinto:

- **Errores**: Informan de problemas que hacen imposible compilar el programa. `gcc` indica dónde esta el error mediante el nombre del fichero y el número de línea donde *cree* que puede estar el problema.
- **Avisos**: Indican un problema, aunque la compilación continúa. Los mensajes de aviso también indican el fichero y el número de línea, pero incluyen el texto `warning` para distinguirlos de los mensajes de error.

Los avisos pueden indicar puntos de peligro que se deberían comprobar para asegurarse que el programa realmente hace lo que debe. Muchos avisos sólo se generan si se incluye la opción `-Wall`. Por ello, se aconseja utilizar el siguiente formato para compilar programas con `gcc`:

```
gcc -o nombre_ejecutable fuente.c [fuente2.c...] -Wall
```

Los avisos no se deberían ignorar, porque normalmente indican que hay algo mal en el programa, y seguramente se comportará de forma diferente a la esperada. Hay otra opción de compilación que comprueba si se están utilizando variables sin inicializar. Dicha opción se activa con la opción `-O`.

Los mensajes de error y los avisos aparecen precedidos por el nombre del fichero y la función donde se ha encontrado el error. Por ejemplo:

```
prog.c: In function ‘main’:
```

indica que el error está en el fichero `prog.c`, y específicamente en la función `main`. Las siguientes líneas indican los errores que ha encontrado dentro de dicha función. Si el programa tiene más de una función o más de un fichero, entonces los errores de cada función y fichero aparecerán listados separadamente. Las líneas de error o aviso que siguen la línea anterior indican el lugar aproximado donde se ha producido el error (en ocasiones el compilador no es capaz de encontrar exactamente la línea errónea). El formato con el que aparecen es el nombre del fichero, seguido por la línea dentro de ese fichero, la palabra `warning` (si es un aviso) y una descripción del problema. Por ejemplo:

```
prog.c:3: warning: unknown escape sequence ‘\z’
```

indica que hay un problema en la línea 3 del fichero `prog.c`. El mensaje es un aviso. El problema encontrado es que el carácter `\z` no es una secuencia de escape conocida. Si no hay más errores, el compilador generará el programa, y se podrá ejecutar, pero seguramente tendrá un comportamiento extraño.

A continuación aparece otro ejemplo de salida de errores:

```
prog.c: In function ‘factorialI’:
prog.c:7: ‘f’ undeclared (first use in this function)
prog.c:7: (Each undeclared identifier is reported only once
prog.c:7: for each function it appears in.)
prog.c: In function ‘combina’:
prog.c:18: ‘retun’ undeclared (first use in this function)
prog.c:18: parse error before ‘i’
prog.c: In function ‘main’:
prog.c:25: parse error before ‘}’
```

En este caso, los errores se reparten entre la función `factorialI` (1 error), `combina` (2 errores) y `main` (1 error).

Ten en cuenta que, en la mayoría de las ocasiones, un error puede inducir otros muchos. A la hora de depurar los errores de un programa, empieza siempre por el primero, y recompila a menudo, ya que los demás pueden desaparecer en cascada.

A continuación se muestra una lista de los errores y avisos más comunes del compilador `gcc`, junto a una pequeña descripción del error. Si encuentras un error no listado en este documento, por favor mándamelo para añadirlo (`fjabad@dsic.upv.es`).

2. Errores comunes en `gcc`

2.1. `'variable' undeclared (first use in this function)`

C es un lenguaje tipado. Esto significa que hay que declarar las variables antes de usarlas. C es sensible a mayúsculas, por lo que las variables `pepe` y `Pepe` son distintas.

Las causas del error pueden ser:

- no se ha declarado la variable
- se ha escrito mal (¡cuidado con las mayúsculas!)

Este error normalmente se acompaña con el mensaje:

```
prog.c:7: (Each undeclared identifier is reported only once
prog.c:7: for each function it appears in.)
```

que indica que, aunque la variable que no se ha declarado se utilice varias veces en la función, sólo se informará de la primera aparición.

2.2. `No such file or directory`

El usuario ha pedido a `gcc` que compile un fichero que no existe. El compilador `gcc` espera que los ficheros fuente tengan la extensión `.c`. Si el fichero fuente se llama `prog.c`, las siguientes llamadas son erróneas:

```
gcc prog
gcc -o prog.c prog
gcc prog -o prog.c
```

Este error puede ir acompañado del mensaje `No input files`. En alguno de los ejemplos anteriores, puede ocurrir que el compilador borre el fichero fuente, perdiendo el programa tecleado. El comando correcto sería:

```
gcc -o prog prog.c -Wall
```

2.3. `parse error before 'string'`

El compilador ha encontrado algo que no entiende. Las sentencias de un programa C empiezan normalmente con una palabra reservada, un nombre de variable o un nombre de función. Este error se genera normalmente cuando no se reconoce la primera palabra de una línea de código:

```
main() {
  into printf("Hola, mundo\n");
}
```

Otras posibles causas pueden ser:

- falta un paréntesis ')' o una llave de cierre '}' en los alrededores de la línea indicada
- hay más llaves de la cuenta
- falta un punto y coma (;) en la línea anterior
- hay una expresión mal construida
- la condición de un `if` o de un `while` no va entre paréntesis

2.4. `missing terminating " character`

No se encuentra el carácter de terminación de la definición de una cadena. Este mensaje suele venir acompañado por un mensaje que indica la posición donde el compilador cree que puede haberse producido el fallo. Dicho mensaje es "`possible start of unterminated string literal`".

Un ejemplo de código que produce este error es:

```
printf("Coord. vértices (x1<=x2, y1<=y2)\n);
```

2.5. `undefined reference to 'función'`

El mensaje de error completo es parecido al siguiente:

```
/tmp/ccabbz60.o: In function 'pepe':
/tmp/ccabbz60.o(.text+0x10): undefined reference to 'juan'
```

`gcc` ha encontrado lo que parece una llamada a una función, pero no encuentra el código de dicha función. El nombre de la función desaparecida es la última palabra de la segunda línea (en el ejemplo anterior, `juan`). En el caso de que la función que falta sea la función `main`, se obtendrá un error como:

```
/usr/lib/gcc-lib/i386-redhat-linux/2.96/../../../crt1.o: In function '_start':
/usr/lib/gcc-lib/i386-redhat-linux/2.96/../../../crt1.o(.text+0x18): undefined reference to 'main'
collect2: ld returned 1 exit status
```

Todos los programas C deben definir una función `main`, porque es la función que se invoca al ejecutar el programa.

Estos errores aparecen principalmente debido a:

- Errores tipográficos. Se ha escrito mal el nombre de la función (¡cuidado con las mayúsculas!)
- La función está en otro fichero. En ese caso, la orden para compilar es:

  ```
  gcc -o prog fuente1.c fuente2.c -Wall
  ```

Si la función que falta es de la librería matemática (`sqrt`, `sin`, `tan`...), el error indica que no se ha enlazado la librería matemática. Para ello, compilar con la opción `-lm`:

```
gcc -o prog fuente1.c fuente2.c -lm -Wall
```

2.6. `incompatible type for argument 1 of 'función'`

Este error aparece cuando se le pasa a una función un argumento con un tipo que no concuerda con el esperado. La principal causa de este error es que no se ha respetado el orden de los argumentos definido por la cabecera de la función.

2.7. `unterminated string or character constant`

Hay una cadena que abre comillas, pero no las cierra. Normalmente la línea indicada no es la que contiene el fallo. Hay que comprobar desde la línea de error hacia atrás que todas las cadenas tienen las comillas de apertura y de cierre. Por ejemplo, en la línea siguiente faltan las comillas de cierre:

```
printf("tienes que cerrar todas las cadenas);
```

2.8. `parse error at end of input`

Se ha producido un error al final del código fuente. Posiblemente falta una llave de cierre en algún lugar del programa. El compilador no puede decir dónde, así que deberás recorrer todo el programa. También puede ser que algún comentario no tenga el código de cierre, es decir, que haya un `/*` que no acabe con un `*/`.

2.9. `invalid macro name`

Nombre de macro no válido. Los nombres de macro deben seguir las mismas reglas que los nombres de variable. La forma de definir una macro es la siguiente:

`#define` *nombre* *texto de reemplazo*

No se pueden introducir saltos de línea en la definición, si no acaba la línea precedente con "\", por ejemplo:

```
#define MACRO_LARGA { z=y; \
                      y=x; \
                      x=z; }
```

3. Avisos comunes en `gcc`

3.1. `unknown escape sequence '\z'`

Las secuencias de escape son un carácter precedido por '\'. Por ejemplo, '`n`' es el carácter `n`, pero '`\n`' es el carácter de retorno de carro. Algunos caracteres, como `z`, no representan una secuencia de escape. El programa seguirá compilando porque este mensaje es un aviso, pero seguramente no se comportará como se espera.

Las causas de este aviso pueden ser:

- Error tipográfico
- Se quiere mostrar el carácter '\' por pantalla. En ese caso, utilizar '\\'.

3.2. `multi-character character constant`

En C, las cadenas se delimitan por comillas dobles (`"cadena"`), y los caracteres o secuencias de escape por comillas simples (`'a'`, o `'\n'`).

3.3. `passing arg n of 'función' makes pointer from integer without a cast`

En la llamada a la función cuyo nombre indica el mensaje de error, el argumento *n-ésimo* se esperaba que fuera un puntero, pero se le ha pasado un entero sin convertir su tipo explícitamente.

C es un lenguaje tipado, así que el tipo de los datos que se pasan a una función y el tipo declarado de los argumentos deben coincidir. Este aviso se recibe siempre que se da una incompatibilidad de tipos entre los datos que se pasan a una función y el tipo que se espera. Este aviso no

se debe ignorar, ya que normalmente indica un error de tipo lógico. El siguiente programa genera este aviso:

```
#include <stdio.h>

int suma(int *a) {
   return (*a)+1;
}

int main() {
   int b=10;
   b=suma(b);
   return 0;
}
```

La causa más común de aparición de este error es que se está pasando una variable de tipo `int`, pero se espera una variable de tipo `int *`. Para solucionarlo, hay que utilizar el operador `&` para obtener el puntero correspondiente. Para el ejemplo anterior:

```
   b=suma(&b);
```

3.4. `implicit declaration of function ‘función’`

Se ha encontrado una llamada a una función antes de conocer su perfil. Esto ocurre normalmente porque la función aparece después de su llamada, o porque está en otro fichero. La solución pasa por llevar la cabecera de la función a un fichero `.h`, e incluirlo al principio de todos los ficheros que llamen a la función. Los ficheros de cabecera tienen la extensión `.h` (como, por ejemplo, `stdio.h`), y contienen las cabeceras de las funciones que un programa ofrece al exterior. Por ejemplo:

```
int suma(int a,int b);
int resta(int a,int b);
int multiplica(int a,int b);
```

Si, por ejemplo, se utilizan las rutinas matemáticas sin incluir la cabecera `math.h`, puede ocurrir que rutinas como `sin`, `cos`, `sqrt`, etc. devuelvan valores extraños. Esto es debido a que el compilador asume por defecto que las funciones que no conoce devuelven un entero. Posteriormente, si la función devolvía otro tipo de datos (por ejemplo, `float`), entonces el programa fallará. La solución es incluir las cabeceras necesarias.

3.5. `suggest parentheses around assignment used as truth value`

El compilador muestra este aviso cuando se encuentra una línea como la siguiente:

```
if (i = 1009)
    printf("Esto se imprime siempre!!!\n");
```

En C, el signo '`=`' indica asignación. Por lo tanto, la línea anterior asigna el valor `1009` a la variable `i`. Según las reglas de C, el resultado de hacer una asignación es el valor de la parte derecha. Por lo tanto, el resultado de `i = 1009` es `1009` que, como es distinto de cero, se considera verdadero y el `if` siempre se cumple. Si se quería utilizar el operador de comparación, cambiar el `=` por `==`.

3.6. `format argument is not a pointer (arg n)`

En una llamada a `scanf`, no se ha pasado el puntero a la variable donde almacenar el valor introducido por el usuario. Por ejemplo, el siguiente programa es incorrecto:

```
int b=10;

scanf("%d",b);
```

Si se ignora el aviso y se ejecuta el programa resultante, seguramente al pasar por esa línea el programa provocará una excepción y acabará.

Para solucionarlo, a `scanf` siempre hay que pasar la dirección de la variable que debe recoger el valor:

```
scanf("%d",&b);
```

3.7. `return type of ‘main’ is not ‘int’`

Toda función `main` de un programa C debe declararse de tipo entero. Se generará este aviso si se declara la función `main` como sigue:

```
void main() {
...
```

La forma correcta de la función `main` es la siguiente:

```
int main() {

[...]

   return 0;
}
```

El valor que devuelve la función `main` es el valor que se pasa al sistema operativo como código de error. Un programa que termina normalmente debe devolver un cero. Cuando un programa devuelve un número distinto de cero indica que no se ha podido completar la ejecución debido a un error (por ejemplo, porque no ha encontrado un fichero).

3.8. `‘return’ with a value, in function returning void`

Este aviso indica que hay una función declarada como `void` que devuelve un valor. Por ejemplo:

```
void funcion(int a) {
   [...]
   return a+1;
}
```

Las funciones declaradas como `void` no pueden devolver valores. Pueden utilizar `return` para acabar, pero sin valor.

3.9. `control reaches end of non-void function`

Este aviso puede aparecer en aquellas funciones que devuelven valores. Indica que hay una forma de ejecutar dicha función sin devolver un valor. Por ejemplo:

```
int funcion(int a) {
   if (a<4) return 2;
}
```

El programador se debe asegurar que su función devuelve algo independientemente del camino que siga la ejecución.

4. Más información...

En las siguientes páginas web puedes obtener más información sobre los errores producidos por gcc:

- http://www.cs.um.edu.mt/~cstaff/courses/lectures/csa2170/c_errors.html
- http://cs-www.bu.edu/help/unix/the_four_most_common_gcc_error_messages.html
- http://web.cs.mun.ca/~michael/c/problems.html