Introducción a la Programación en C

La función `main()`

En esta sección, empezaremos con el comienzo de cada programa en C, la función `main()`. Sin embargo, primero hablemos filosóficamente sobre qué es una función. Desde una perspectiva de programación, las funciones permiten agrupar una serie lógica de actividades, o *declaraciones de programa*, bajo un nombre. Por ejemplo, supongamos que quiero crear una función llamada `bakeCake`. Mi algoritmo para hornear un pastel podría verse así:

> Mezclar los ingredientes húmedos en un bol
>
> Combinar los ingredientes secos
>
> Verter la masa con una cuchara en una bandeja para hornear engrasada
>
> Hornear el pastel a 350 grados durante 30 minutos

Cualquiera que lea mi código verá mi función llamada `bakeCake` y sabrá de inmediato que estoy tratando de hornear pasteles.

Las funciones normalmente no son estáticas, lo que significa que son entidades vivas y que respiran, nuevamente filosóficamente, que toman y devuelven información. Por lo tanto, mi función `bakeCake` tomaría una lista de ingredientes para hornear (llamados *parámetros*) y devolvería un pastel terminado (llamado *valor*).

> **Algoritmos**
>
> Un algoritmo es un proceso finito paso a paso para resolver un problema. Puede ser tan simple como una receta para hornear un pastel o tan complicado como el proceso para implementar un sistema de piloto automático para un avión jumbo 747.
>
> Los algoritmos generalmente comienzan con un enunciado del problema. Es este enunciado del problema que los programadores usan para formular el proceso para resolver el problema. Tenga en cuenta que el proceso de creación y análisis de algoritmos ocurre antes de que se haya escrito cualquier código de programa.

La función `main()` es como cualquier otra función de programación en el sentido de que agrupa actividades similares y puede tomar parámetros (información) y devolver valores (nuevamente, información). Sin embargo, lo que hace que la función `main()` sea única con respecto a otras funciones es que los valores que devuelve se devuelven al sistema operativo. Otras funciones que usará y creará en este libro devuelven valores a la declaración C que realiza la llamada dentro de la función `main()`.

En este libro, usaré funciones `main()` que no tienen parámetros (funciones que no toman parámetros) y no devuelven valores (`void`).

```
void main()

{

}
```

Como muestra el ejemplo anterior, la función `main()` comienza con la palabra clave `main` y va seguida de dos paréntesis vacíos `()`. Los paréntesis se utilizan para incluir los parámetros que se pasarán a la función `main()`. `void` indica que la función no devolverá valor alguno al sistema.

C es un lenguaje de programación que distingue entre mayúsculas y minúsculas. Por ejemplo, los nombres de las funciones `main()`, `Main()` y `MAIN()` no son iguales. Se necesitan recursos informáticos adicionales para NO distinguir entre mayúsculas y minúsculas, ya que los dispositivos de entrada, como los teclados, distinguen entre mayúsculas y minúsculas.

Después de los paréntesis hay dos llaves. La primera llave indica el comienzo de un bloque de programación lógica y la última llave indica el final de un bloque de programación lógica. Cada implementación de función requiere que utilice una llave de inicio, {, y una llave de cierre, }.

El siguiente código de programa demuestra un programa C completo y simple. A partir de este código, aprenderá cómo las declaraciones de un solo programa se unen para formar un programa C completo.

```
/* Programación en C para principiantes */
// por Michael Vine

#include <stdio.h>

void main()
{
    printf("\nC you later\n");
}
```

Cuando se compila y se ejecuta el programa anterior, muestra el texto "*C you later*" en la pantalla de la computadora, como se muestra en la Figura 1.2.

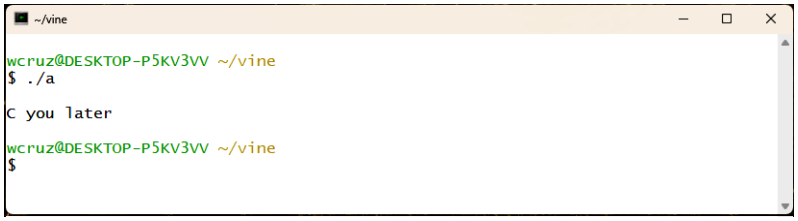

Fig. 1.2 Programa en C con salida estándar.

Revise el código del programa de muestra en la Figura 1.3; podrá ver los numerosos componentes que componen un pequeño programa en C.

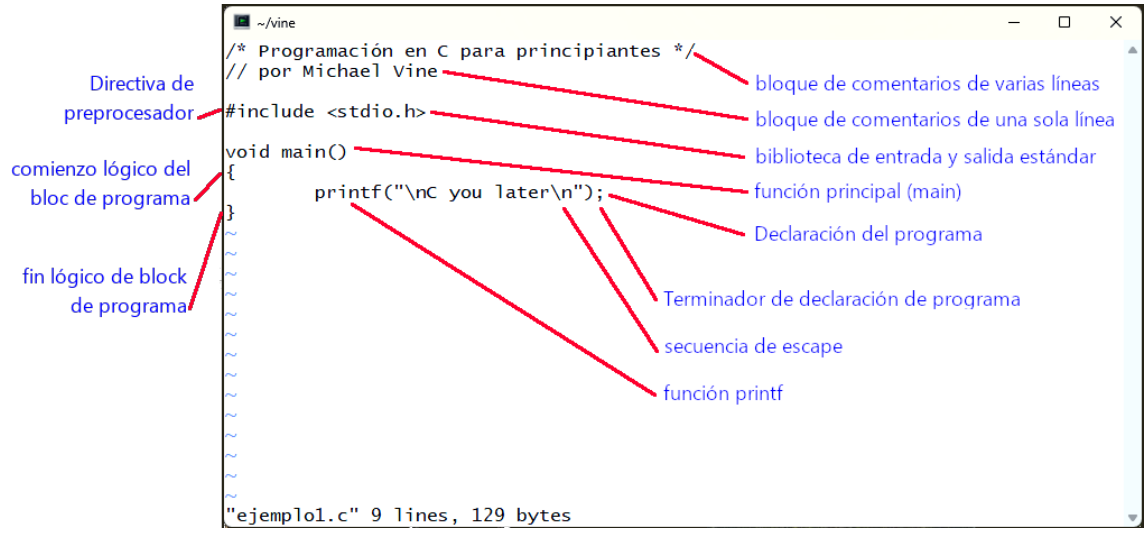

Fig. 1.3 Bloques de construcción de un programa C simple.

En este capítulo cubriremos estos componentes y cómo se utiliza cada uno para construir un programa C simple.

Comentarios

Los *comentarios* son una parte integral del código del programa en cualquier lenguaje de programación. Los comentarios ayudan a identificar el propósito del programa y a explicar rutinas complejas. Pueden ser valiosos para usted como programador y para otros programadores que vean su código.

En la siguiente línea de código, el compilador ignora el texto *Programación en C para principiantes* porque está rodeado por los conjuntos de caracteres /* y */.

```
/* Programación en C para principiantes */
```

El conjunto de caracteres /* indica el comienzo de un bloque de comentarios; el conjunto de caracteres */ identifica el final de un bloque de comentarios. No es necesario que estos conjuntos de caracteres estén en la misma línea y se pueden utilizar para crear comentarios tanto de una sola línea como de varias líneas. Para demostrarlo, el siguiente bloque de código muestra la utilidad de los comentarios de varias líneas.

```
/* Programación en C para principiantes
   Capítulo 1: Introducción a la programación en C
   Por Michael Vine
*/
```

Es posible que su programa en C no se compile correctamente si omite uno de los conjuntos de caracteres de comentario o si invierte los caracteres. Por ejemplo, el siguiente segmento de código omite un conjunto de caracteres de comentario y no se compilará.

```
/* Programación en C para principiantes
```

La siguiente línea de código tampoco se compilará porque los conjuntos de caracteres de comentario se ordenaron incorrectamente.

```
*/ Programación en C para principiantes */
```

También puede crear comentarios rápidos de una línea con el conjunto de caracteres //. La siguiente línea de código lo demuestra.

```
//por Michael Vine
```

Si su compilador de C admite C++, lo que hace gcc, puede usar el conjunto de caracteres de una sola línea // para comentarios de una línea. Aunque es poco probable, tenga en cuenta que no todos los compiladores de C admiten el conjunto de caracteres de una sola línea.

El compilador ignora todos los caracteres que se leen después del conjunto de caracteres // solo para esa línea. Para crear un bloque de comentarios de varias líneas con el conjunto de caracteres //, necesitará los caracteres de comentario al frente de cada línea. Por ejemplo, el siguiente código crea un bloque de comentarios de varias líneas.

```
//Programación en C para principiantes
//Capítulo 1: Introducción a la programación en C
//Por Michael Vine
```

Palabras clave

Hay 32 palabras definidas como palabras clave en el lenguaje de programación C estándar ANSI. Estas palabras clave tienen usos predefinidos y no se pueden utilizar para ningún otro propósito en un programa en C. El compilador, en este caso gcc, las utiliza como ayuda para crear el programa. Tenga en cuenta que estas palabras clave siempre deben escribirse en minúsculas (consulte la Tabla 1.1).

TABLA 1.1 PALABRAS CLAVE DEL LENGUAJE C

Palabra clave	Descripción	Palabra clave	Descripción
`auto`	Define una variable local como si tuviera una duración local	`int`	Tipo de datos básico
`break`	Pasa el control fuera de la construcción de programación	`long`	Modificador de tipo
`case`	Control de bifurcación	`register`	Almacena la variable declarada en un registro de CPU
`char`	Tipo de datos básico	`return`	Sale de la función
`const`	Valor no modificable	`short`	Modificador de tipo
`continue`	Pasa el control al comienzo del bucle	`signed`	Modificador de tipo
`default`	Control de bifurcación	`sizeof`	Devuelve expresión o tamaño de tipo
`do`	Bucle Do While	`static`	Conserva el valor de la variable después de que finaliza su alcance
`double`	Tipo de datos de punto flotante	`struct`	Agrupa las variables en un solo registro
`else`	Instrucción condicional	`switch`	Control de bifurcación
`enum`	Define un grupo de constantes de tipo `int`	`typedef`	Crea un nuevo tipo
`extern`	Indica un identificador como se define en otra parte	`union`	Agrupa las variables que ocupan el mismo espacio de almacenamiento
`float`	Tipo de datos de punto flotante	`unsigned`	Modificador de tipo
`for`	Bucle For	`void`	Tipo de datos vacío
`goto`	Transfiere el control del programa incondicionalmente	`volatile`	Permite que una rutina en segundo plano cambie una variable
`if`	Instrucción condicional	`while`	Repite la ejecución del programa mientras la condición sea verdadera

Tenga en cuenta que, además de la lista de palabras clave anteriores, Es posible que su compilador de lenguaje C defina algunas más. Si es así, se incluirán en la documentación que acompaña al compilador.

A medida que avance en este libro, le mostraré cómo utilizar muchas de las palabras clave del lenguaje C mencionadas anteriormente.

Declaraciones de Programa

Muchas líneas de los programas en C se consideran declaraciones de programa, que sirven para controlar la ejecución y la funcionalidad del programa. Muchas de estas declaraciones de programa deben terminar con un terminador de declaración. Los terminadores de declaración son simplemente puntos y coma (;). La siguiente línea de código, que incluye la función `printf()`, muestra una declaración de programa con un terminador de declaración.

```
printf("\nC you later\n");
```

Algunas declaraciones de programa comunes que no requieren el uso de terminadores de declaración

son las siguientes:

- Comentarios
- Directivas del preprocesador (por ejemplo, `#include` o `#define`)
- Identificadores de bloque de inicio y fin del programa
- Inicios de definición de función (por ejemplo, `main()`)

Las declaraciones de programa anteriores no requieren el terminador de punto y coma (;) porque no son declaraciones de C ejecutables ni llamadas de función. Solo las declaraciones de C que realizan trabajo durante la ejecución del programa requieren los puntos y coma.

Una función que se usa comúnmente para mostrar la salida en la pantalla de la computadora es la función `printf()`. Como se muestra a continuación, la función `printf()` se utiliza para escribir el texto "*C you later*" en la salida estándar (mostrado en la Figura 1.2).

```
printf("\nC you later\n");
```

Como la mayoría de las funciones, la función `printf()` toma un valor como parámetro. (Hablaré más sobre las funciones en el Capítulo "Programación estructurada"). Cualquier texto que desee mostrar en la salida estándar debe estar entre comillas.

En la mayoría de los casos, los caracteres o el texto que desea que aparezcan en la pantalla se colocan entre comillas, con la excepción de los caracteres de escape o las secuencias de escape. El carácter de barra invertida (\) es el carácter de escape. Cuando se ejecuta la instrucción `printf()` que se muestra arriba, el programa espera el siguiente carácter que sigue a la barra invertida. En este caso, el siguiente carácter es el carácter `n`. Juntos, la barra invertida (\) y los caracteres `n` forman una secuencia de escape.

Secuencias de escape

Las secuencias de escape son caracteres especialmente secuenciados que se utilizan para dar formato a la salida.

Esta secuencia de escape en particular (\n) le indica al programa que agregue una nueva línea. Observe la siguiente declaración del programa. ¿Cuántas líneas nuevas se agregan a la salida estándar con esta función `printf()`?

```
printf("\nC you later\n");
```

Esta función `printf()` agrega dos líneas nuevas para fines de formato. Antes de que se muestre cualquier texto, el programa genera una nueva línea. Después de que el texto se escribe en la salida estándar, en este caso la pantalla de la computadora, se escribe otra línea nueva.

La Tabla 1.2 describe algunas secuencias de escape comunes.

TABLA 1.2 SECUENCIAS DE ESCAPE COMUNES

Secuencia de escape	Propósito
\n	Crea una nueva línea
\t	Mueve el cursor a la siguiente pestaña
\r	Mueve el cursor al principio de la línea actual
\\	Inserta una barra invertida
\"	Inserta una comilla doble
\'	Inserta una comilla simple

Secuencia de escape \n

Como se muestra en las Figuras 1.4 y 1.5, la secuencia de escape \n se puede utilizar de múltiples maneras para dar formato a la salida.

El siguiente segmento de código genera tres líneas separadas con solo una función `printf()`.

```
printf("line 1\nline2\nline3\n");
```

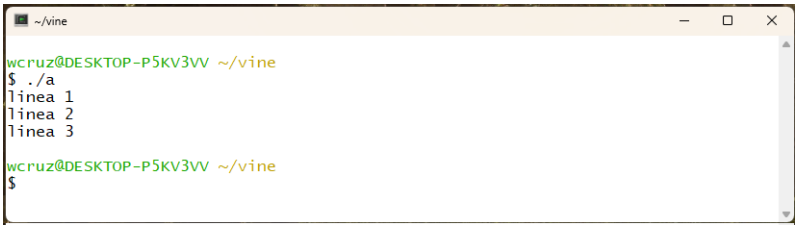

FIGURA 1.4 Uso de la secuencia de escape \n con una función `printf()` para generar múltiples líneas.

El siguiente segmento de código demuestra cómo se puede utilizar la secuencia de escape \n con múltiples instrucciones `printf()` para crear una sola línea de salida.

```
printf("C ");
printf("para ");
printf("principiantes\n");
```

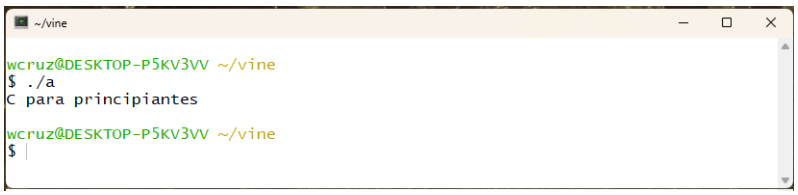

FIGURA 1.5 Uso de la secuencia de escape \n con múltiples funciones `printf()` para generar una sola línea.

Secuencia de escape \t

La secuencia de escape \t mueve el cursor al siguiente espacio de tabulación. Esta secuencia de escape es útil para formatear la salida de muchas maneras. Por ejemplo, un deseo de formato común es crear columnas en la salida, como lo demuestran las siguientes instrucciones del programa.

```
printf("\nDom\tLun\tMar\tMie\tJue\tVie\tSab\n");
printf("\t\t\t\t1\t2\t3\n");
printf("4\t5\t6\t7\t8\t9\t10\n");
printf("11\t12\t13\t14\t15\t16\t17\n");
printf("18\t19\t20\t21\t22\t23\t24\n");
printf("25\t26\t27\t28\t29\t30\t31\n");
```

Como se muestra en la Figura 1.6, las instrucciones del programa anterior crean columnas formateadas que muestran un mes calendario de muestra.

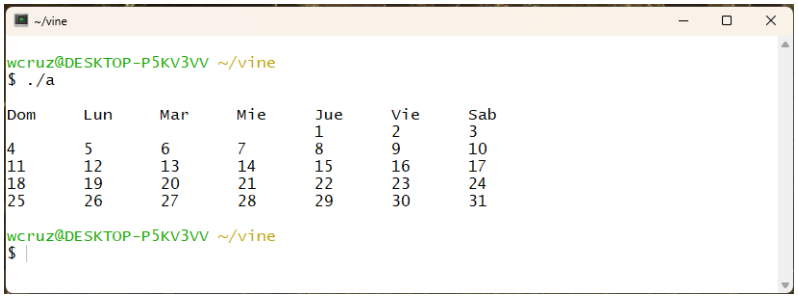

FIGURA 1.6 Demostración del uso de espacios de tabulación y columnas con la secuencia de escape \t.

Secuencia de escape \r

Puede encontrar útil la secuencia de escape \r para algunas tareas de formato cuando la posición del cursor es importante, especialmente con la salida impresa porque una impresora puede sobrescribir el texto que ya se está imprimiendo. El siguiente código de programa demuestra cómo funciona.

```
printf("Esta secuencia de escape mueve el cursor ");
printf("al principio de esta línea\r");
```

Secuencia de escape \\

La secuencia de escape \\ inserta una barra invertida en el texto. Esto puede parecer innecesario al principio, pero recuerde que siempre que el programa lee una barra invertida en una función `printf()`, espera ver un carácter de escape válido inmediatamente después. En otras palabras, el carácter de barra invertida (\) es un carácter especial en la función `printf()`; si necesita mostrar una barra invertida en el texto, debe utilizar esta secuencia de escape. La siguiente declaración del programa muestra la secuencia de escape \\.

```
printf("c:\\cygwin\\bin debe estar en la ruta de su sistema");
```

Secuencia de escape \"

Otro carácter reservado en la función `printf()` es el carácter de comilla doble ("). Para insertar una comilla en el texto de salida, utilice la secuencia de escape \" como se muestra en la siguiente declaración del programa.

```
printf("\"Este es un texto entre comillas\"");
```

Secuencia de escape \'

Similar a la secuencia de escape de comillas dobles (\") es la secuencia de escape de comillas simples (también llamada apóstrofo) (\'). Para insertar una comilla simple en el texto de salida, utilice la secuencia de escape \' como se muestra en la siguiente declaración del programa.

```
printf("\nUna comilla simple se ve como \'\n");
```

Directivas

A continuación, se muestra otro vistazo al programa de muestra que se mostró anteriormente en el capítulo.

```
/* Programación en C para principiantes absolutos */
//por Michael Vine

#include <stdio.h>

void main()
{
    printf("\nC you later\n");
}
```

Observe la declaración del programa que comienza con el signo de almohadilla (#):

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

Cuando el preprocesador de C encuentra el signo de almohadilla, realiza ciertas acciones según la directiva que se produce antes de la compilación. En el ejemplo anterior, le dije al preprocesador que incluyera la biblioteca `stdio.h` con mi programa.

El nombre `stdio.h` es la abreviatura de *archivo de encabezado de entrada y salida estándar*. Contiene enlaces a varias funciones de biblioteca estándar de C, como `printf()`. Excluir esta directiva del preprocesador no tendrá un efecto adverso al compilar o ejecutar su programa. Sin embargo, incluir el archivo de encabezado permite que el compilador lo ayude mejor a determinar las ubicaciones de los errores. Siempre debe agregar una directiva para incluir cualquier archivo de encabezado de biblioteca que use en sus programas de C.

Más adelante, aprenderá otras funciones comunes de la biblioteca, cómo usar otras directivas de preprocesador como macros y cómo crear sus propios archivos de biblioteca.

Compilador GCC

El compilador gcc es un compilador de C estándar ANSI. Un programa C pasa por muchos pasos antes de convertirse en un programa en ejecución. El compilador `gcc` realiza varias tareas por usted. Las más notables son las siguientes:

- Preprocesa el código del programa y busca varias directivas.
- Genera códigos de error y mensajes, si corresponde.
- Compila el código del programa en un código objeto y lo almacena temporalmente en el disco.
- Vincula cualquier biblioteca necesaria al código objeto y crea un archivo ejecutable y lo almacena en el disco.

> **ANSI**
>
> ANSI es una abreviatura de *American National Standard for Information Systems*. El objetivo común de ANSI es proporcionar estándares informáticos para las personas que usan sistemas de información.

Use la extensión `.c` al crear y guardar programas C. Esta extensión es la convención de nombres estándar para programas creados en C. Para crear un nuevo programa en C, invoque un editor de texto como `nano` o `VIM` como se muestra a continuación.

```
nano hello.c
```

```
vim hello.c
```

`nano` es otro editor de texto común basado en UNIX que viene con el paquete de software Cygwin. Desde la perspectiva del usuario final, es mucho más intuitivo y fácil de usar que VIM, pero no tiene la cantidad de funcionalidad que VIM. Aunque no se selecciona en una instalación predeterminada de Cygwin, nano y otros editores de texto se pueden seleccionar durante la instalación a través de la ventana Seleccionar paquetes.

Las dos instrucciones de comando anteriores abren un editor de texto y crean un nuevo archivo llamado `hello.c`

Una vez que haya creado un programa en C utilizando un editor, como `nano` o `VIM`, estará listo para compilar su programa utilizando `gcc`.

Desde el shell de UNIX Cygwin, escriba lo siguiente:

```
gcc hello.c
```

Si su programa se compila correctamente, `gcc` creará un nuevo archivo ejecutable llamado `a.exe`

Si no puede ejecutar su programa compilado, entonces debe verificar la estructura de directorio `%drive%:\cygwin\bin` (donde `%drive%` es la letra de la unidad donde está instalado Cygwin) se haya agregado a la variable de ruta (`path`) del sistema.

`a.exe` es el nombre predeterminado para todos los programas C compilados con esta versión de gcc. Si está programando con una versión diferente de gcc en un sistema operativo UNIX, el nombre del archivo puede ser `a.out`

Cada vez que compila un programa C con gcc, sobrescribe los datos anteriores contenidos en el archivo `a.exe`. Puede corregir esto proporcionando a gcc una opción para especificar un nombre único para su archivo ejecutable. La sintaxis para especificar un nombre único para el ejecutable es la siguiente.

```
gcc programName -o executableName
```

La palabra clave `programName` es el nombre de su programa C, la opción `-o` (letra o) le indica a gcc que especificará un nombre de compilación único y la palabra clave `executableName` es el nombre de salida deseado. Aquí hay otro ejemplo que utiliza nombres de archivo reales.

```
gcc hello.c -o hello.exe
```

Puede encontrar una gran cantidad de información sobre el programa gcc accediendo a las páginas `man` de gcc (las páginas manuales en línea para comandos UNIX) desde el indicador UNIX como se muestra aquí.

```
man gcc
```

Para ejecutar su programa desde el indicador de Cygwin UNIX, escriba lo siguiente:

```
./hello
```

A diferencia de Windows, el shell de UNIX no busca de forma predeterminada en el directorio actual cuando intenta ejecutar un programa. Al anteponer al nombre de su programa compilado la secuencia de caracteres `./`, le está indicando al shell de UNIX que busque el programa C compilado, en este caso hola, en el directorio actual.

Si está utilizando un sistema Microsoft Windows, también puede ejecutar su programa desde un shell de comandos basado en Microsoft, a menudo denominado indicador de DOS (siempre que esté en

el directorio de trabajo), simplemente escribiendo el nombre del programa.

Tenga en cuenta que en ambos casos no es necesario seguir el nombre del programa compilado con la extensión de archivo `.exe`

Cómo depurar programas C

Si su programa se compila, sale o se ejecuta de forma anormal, es casi seguro que hay un error (un fallo) en su programa. Una buena parte de su tiempo de programación se dedicará a encontrar y eliminar estos errores. Esta sección proporciona algunos consejos para ayudarlo a comenzar. Sin embargo, recuerda que la depuración es tanto un arte como una ciencia informática y, por supuesto, cuanto más practiques la programación, más fácil te resultará.

A menudo, un programa se compila y se ejecuta sin problemas, pero con resultados inesperados. Por ejemplo, el siguiente programa y su salida, que se muestra en la Figura 1.11, se compila y se ejecuta sin errores, pero la salida es ilegible o, en otras palabras, no es lo que esperaba.

```
#include <stdio.h>

void main()
{
    printf("Capítulo 1 - Introducción a la programación en C");
    printf("Este es un ejemplo de un error de formato..");
    printf("El problema de formato se puede corregir utilizando");
    printf("Las secuencias de escape \n y \\ ");
}
```

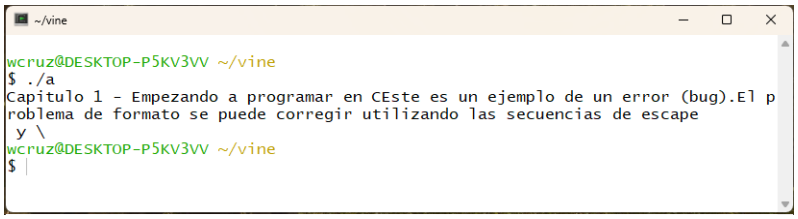

FIGURA 1.11 Ejemplo de error de formato.

¿Puede ver dónde están los problemas de formato? ¿Qué falta y dónde se deben colocar las correcciones? El siguiente bloque de código y su salida en la Figura 1.12 corrige los problemas de formato con secuencias de escape ubicadas adecuadamente.

```
#include <stdio.h>

void main()
{
    printf("Capítulo 1 - Introducción a la programación en C\n");
    printf("Este es un ejemplo de un error de formato.\n");
    printf("El problema de formato se puede corregir utilizando");
    printf("las secuencias de escape \\n y \\\\");
}
```

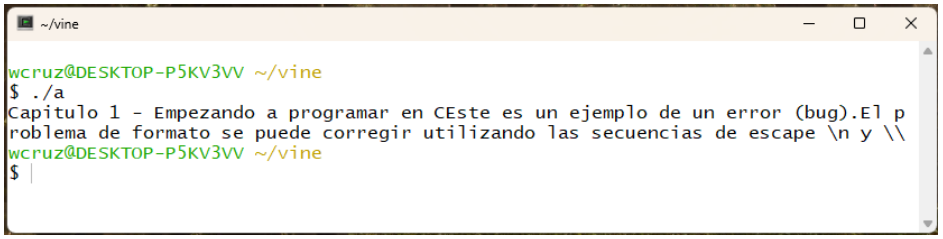

FIGURA 1.12 Corrección de errores de formato con secuencias de escape $\backslash n$ y \\ ubicadas apropiadamente.

Los problemas de formato son comunes en la programación inicial y normalmente se resuelven rápidamente practicando la función `printf()` y las distintas secuencias de escape.

Otro tipo de error común es un error lógico, que incluye un bucle que no sale cuando se espera, una ecuación matemática incorrecta o quizás una prueba de igualdad defectuosa (condición). El primer paso para depurar un error lógico es encontrar la primera línea donde existe el error del programa. Una forma de hacerlo es a través de declaraciones de impresión, utilizando la función `printf()`, esparcidas por todo el código. Por ejemplo, podría hacer algo como esto en su código fuente:

```
anyFunction(int x, int y)
{
    printf("Entering anyFunction()\n"); fflush(stdout);
    ---- lots of your code here ------
    printf("Exiting anyFunction()\n"); fflush(stdout);
}
```

La función `fflush()` garantiza que la declaración de impresión se envíe a su pantalla inmediatamente, y debería utilizarla si está utilizando `printf()` para fines de depuración. El parámetro `stdout` que se pasa a la función `fflush()` es la salida estándar, generalmente la pantalla de la computadora.

Después de haber acotado la línea o función donde se produce el error lógico, el siguiente paso es averiguar el valor de sus variables en ese momento. También puede utilizar la función `printf()` para imprimir valores de variables, lo que le ayudará en gran medida a determinar la fuente del comportamiento anormal del programa. La visualización de valores de variables mediante la función `printf()` se analizará más adelante.

Recuerde, después de corregir cualquier error, debe volver a compilar su programa, ejecutarlo y depurarlo nuevamente si es necesario.

Los programadores principiantes, con mayor frecuencia, encontrarán errores de compilación en lugar de errores lógicos, que generalmente son el resultado de problemas de sintaxis, como identificadores y terminadores faltantes o directivas, secuencias de escape y bloques de comentarios no válidos.

La depuración de errores de compilación puede ser una tarea abrumadora, especialmente cuando se ven 50 o más errores en la pantalla de la computadora. Una cosa importante que hay que recordar es que un solo error en la parte superior del programa puede provocar errores en cascada durante el tiempo de compilación. Por lo tanto, no hace falta decir que el mejor lugar para comenzar a depurar errores de compilación es con el primer error de la lista. En las siguientes secciones, explorará algunos de los errores de compilación más comunes que experimentan los programadores principiantes de C.

Error común #1: falta de identificadores de bloque de programa

Si olvida insertar un identificador de bloque de programa inicial o final correspondiente ({ o }), verá mensajes de error similares a los de la Figura 1.13. En el siguiente ejemplo, omití intencionalmente utilizar el identificador de bloque de programa inicial ({) después del nombre de la función `main()`.

```
#include <stdio.h>

void main()

    printf("Bienvenido a la programación en C\n");
}
```

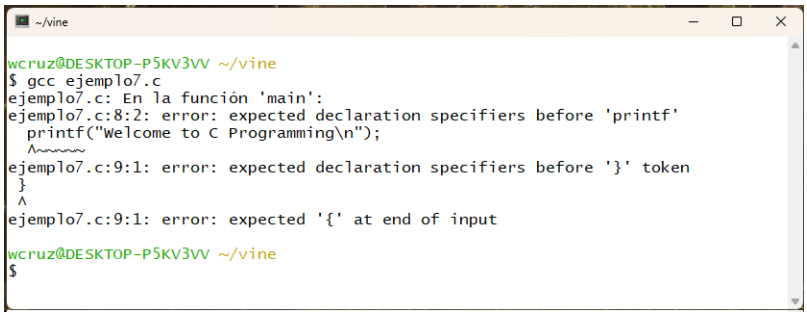

FIGURA 1.13 Identificadores de bloque de programa faltantes.

¡Vaya! La Figura 1.13 muestra muchos errores por el simple hecho de olvidarse de utilizar el identificador de bloque de programa inicial ({). Al depurar errores de compilación, recuerde comenzar simplemente con el primer error, que se muestra a continuación, que me indica que tengo un error justo antes de la función `printf()`. Verá que después de resolver el primer error, muchos de los errores restantes ya no existen.

```
hello.c:8: error: parse error before "printf"
```

Otra pista que le ayudará es mirar el número de línea de la declaración del programa a la que se hace referencia en el error de compilación. En este caso, es la línea número ocho, `hello.c:8:`, que es el número de línea de la función `printf()` en cuestión. Es importante reconocer que el problema *no* está en la declaración de impresión, sino que, como sugiere el error de compilación, existe un problema antes de ella.

Error común #2: terminadores de declaración faltantes

La Figura 1.13 muestra un mensaje de error común generado por algunos escenarios comunes. Este tipo de error de análisis puede generarse por un par de razones. Además de la falta de identificadores de bloques de programa, los errores de análisis pueden ocurrir debido a la falta de terminadores de sentencia (punto y coma).

La figura 1.14 muestra un error en el siguiente programa. ¿Puedes ver dónde existe el error?

```
#include <stdio.h>

void main()

{
    printf("Bienvenido a la programación en C\n")
}
```

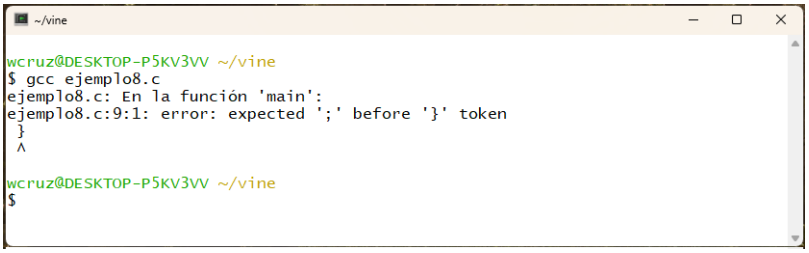

FIGURA 1.14 Sentencias de programa con terminadores faltantes.

Los errores de análisis ocurren porque el compilador de C no puede determinar el final de una sentencia de programa, como la sentencia `print`. En el ejemplo que se muestra en la Figura 1.14, el compilador de C (`gcc`) nos dice que en la línea 10 existe un error de análisis antes de la llave de cierre.

Error común #3: Directivas de preprocesador no válidas

Si escribe una directiva de preprocesador no válida, como escribir mal el nombre de una biblioteca, recibirá un mensaje de error similar al de la Figura 1.15.

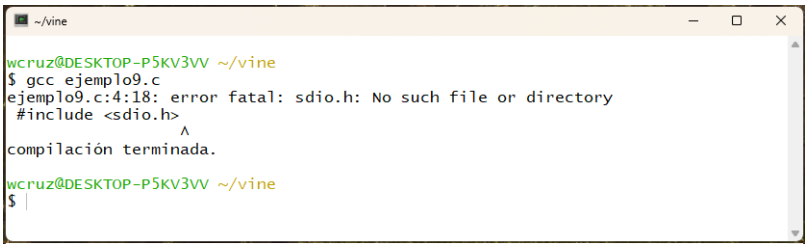

FIGURA 1.15 Nombres de biblioteca mal escritos.

El siguiente bloque de programa con un nombre de biblioteca mal escrito en la directiva del preprocesador causó el error generado en la Figura 1.15. ¿Puedes ver el error?

```
#include <sdio.h>

void main()

{
    printf("Bienvenido a la programación en C\n");
}
```

Este error se produjo porque el archivo de biblioteca `sdio.h` no existe. El nombre de la biblioteca para la entrada y salida estándar debe escribirse `stdio.h`

Error común #4: secuencias de escape no válidas

Al utilizar secuencias de escape, es común utilizar caracteres no válidos o secuencias de caracteres no válidas. Por ejemplo, la Figura 1.16 muestra un error generado por una secuencia de escape no válida.

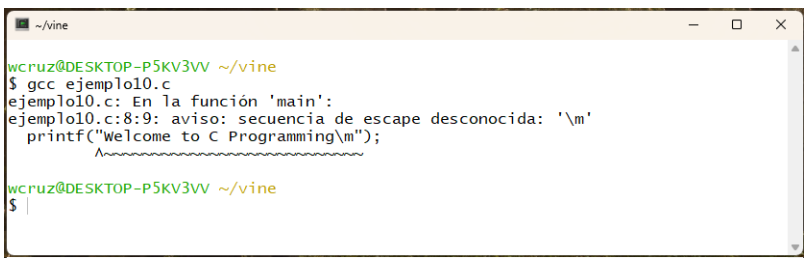

FIGURA 1.16 Secuencias de escape no válidas.

Como se muestra en la Figura 1.16, el compilador gcc es más específico acerca de este error. En concreto, indica que el error está en la línea 7 y que se trata de una secuencia de escape desconocida.

¿Puede identificar la secuencia de escape no válida en el siguiente programa?

```
#include <stdio.h>

void main()

{
    printf("Bienvenido a la programación en C\m");
}
```

Reemplazar la secuencia de escape no válida $\backslash m$ por una secuencia válida como $\backslash n$ corregirá el problema.

Error común #5: bloques de comentarios no válidos

Como se mencionó anteriormente en la sección de comentarios de este capítulo, los bloques de comentarios no válidos pueden generar errores de compilación, como se muestra en la Figura 1.17.

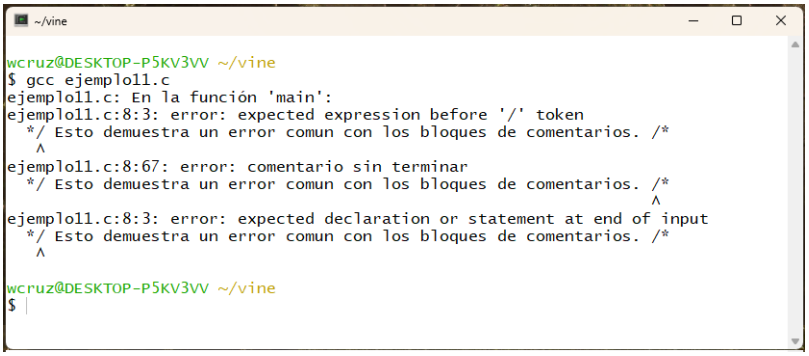

FIGURA 1.17 Errores generados por bloques de comentarios no válidos.

```
#include <stdio.h>

void main()

{
    */ Esto demuestra un error común con los bloques de comentarios. /*
    printf("Bienvenido a la programación en C\n");
}
```

Una corrección simple en el bloque de comentarios, que se muestra a continuación, resolverá el problema y permitirá que el programa se compile correctamente.

```
/* Esto corrige el error del bloque de comentarios anterior */
```

Resumen

- Las funciones le permiten agrupar una serie lógica de actividades, o declaraciones de programa, bajo un nombre.
- Las funciones pueden recibir y devolver información.

- Un algoritmo es un proceso finito paso a paso para resolver un problema.
- Cada implementación de función requiere que utilice una llave de inicio ({) y una llave de cierre (}).
- Los comentarios ayudan a identificar el propósito del programa y a explicar rutinas complejas.
- El conjunto de caracteres /* significa el comienzo de un bloque de comentarios y el conjunto de caracteres */ identifica el final de un bloque de comentarios.
- Hay 32 palabras definidas como palabras clave en el lenguaje de programación estándar ANSI C; estas palabras clave tienen usos predefinidos y no se pueden usar para ningún otro propósito en un programa C.
- La mayoría de las declaraciones de programa controlan la ejecución y la funcionalidad del programa y pueden requerir un terminador de declaración de programa (;).
- Las declaraciones de programa que no requieren un terminador incluyen directivas de preprocesador, bloques de comentarios y encabezados de función.
- La función `printf()` se utiliza para mostrar la salida en la pantalla de la computadora.
- Cuando se combina con la barra invertida (\), los caracteres especiales como `n` forman una secuencia de escape.
- El nombre de la biblioteca `stdio.h` es la abreviatura de entrada y salida estándar y contiene enlaces a varias funciones de la biblioteca C estándar, como `printf()`.
- Los compiladores de C como gcc preprocesan el código del programa, generan códigos de error y mensajes si corresponde, compilan el código del programa en código objeto y vinculan las bibliotecas necesarias.
- Los errores de compilación generalmente son el resultado de problemas de sintaxis, incluidos identificadores y terminadores faltantes, o directivas, secuencias de escape y bloques de comentarios no válidos.
- Un solo error en la parte superior de su programa puede causar errores en cascada durante el tiempo de compilación.
- El mejor lugar para comenzar a depurar errores de compilación es con el primer error.