# 2

# Introducción a la programación en C

### Objetivos

- Ser capaz de escribir programas simples de computación en C.
- Ser capaz de utilizar enunciados simples de entrada y de salida.
- Familiarizarse con los tipos fundamentales de datos.
- Comprender los conceptos de la memoria de la computadora.
- Ser capaz de utilizar operadores aritméticos.
- Comprender la precedencia de los operadores aritméticos.
- Ser capaz de escribir enunciados simples de toma de decisiones.

*¿Qué hay en tu nombre? ¡Lo que llamamos rosa exhalaría el mismo grato perfume con cualquiera otra denominación!*
William Shakespeare
Romeo y Julieta

*Seguí el curso normal ... el que marcan las diferentes ramas de la aritmética la ambición, la confusión, la fealdad y la mofa.*
Lewis Carroll

*Los precedentes que establecen en forma deliberada los sabios deben ponderarse con detenimiento.*
Henry Clay

## Sinopsis



*Resumen • Terminología • Errores comunes de programación • Prácticas sanas de programación • Sugerencia de portabilidad • Ejercicios de autoevaluación • Respuestas a los ejercicios de autoevaluación • Ejercicios.*

## 2.1 Introducción

El lenguaje C facilita un método estructurado y disciplinado para el diseño de programas de computación. En este capítulo presentamos la programación en C, dando varios ejemplos que ilustran muchas características importantes de C. Cada ejemplo se analiza de forma cuidadosa, enunciado por enunciado. En los capítulos 3 y 4 presentaremos una introducción a la *programación estructurada* en C. A partir de ahí el método estructurado será utilizado a todo lo largo del resto del texto.

## 2.2 Un programa simple en C: Cómo imprimir una línea de texto

C utiliza algunas notaciones que pudieran parecer raras a personas que no han programado computadoras. Empezamos analizando un programa simple en C. Nuestro primer ejemplo imprime una línea de texto. El programa y la correspondiente salida en pantalla del programa, se muestran en la figura 2.1.

Aun cuando este programa es simple, ilustra varias características importantes del lenguaje C. Ahora estudiemos en detalle cada línea del programa.

```
/* A first program in C */

main()
{
   printf("Welcome to C!\n");
}
```

```
Welcome to C!
```

**Fig. 2.1** Programa para imprimir texto.



```
/* A first program in C */
```

empieza con `/*` y termina con `*/`, indicando que esta línea es un *comentario*. Los programadores insertan comentarios para *documentar* los programas y mejorar la legibilidad de los mismos. Al ejecutarse el programa, los comentarios no hacen que la computadora realice ninguna acción. Los comentarios serán ignorados por el compilador de C y no harán que se genere ningún código objeto en lenguaje máquina. El comentario `A first program in C` simplemente describe el objetivo del programa. Los comentarios también ayudan a otras personas a leer y comprender su programa, pero demasiados comentarios podrían hacer que un programa sea difícil de leer.

***Error común de programación 2.1***

*Olvidar terminar un comentario con */.*

***Error común de programación 2.2***

*Iniciar un comentario con los caracteres */ o terminar un comentario con los caracteres /*.*

La línea

```
main()
```

forma parte de todo programa de C. Los paréntesis después de `main` indican que `main` es un bloque constructivo del programa conocido como una *función*. Los programas en C contienen una o más funciones, una de las cuales deberá de ser `main`. Todos los programas en C empiezan a ejecutarse en la función `main`.

***Práctica sana de programación 2.1***

*Todas las funciones deberán ser precedidas por un comentario que describa el objeto de la función.*

La *llave izquierda* `{`, debe de iniciar el *cuerpo* de cada función. Una *llave derecha* correspondiente debe dar por terminada cada función. Este par de llaves, y la porción de programa existente entre ambas, también se conoce como un *bloque*. El bloque es una importante unidad de programa en C.

La línea

```
printf("Welcome to C!\n");
```

instruye a la computadora para que ejecute una *acción*, es decir que imprima en la pantalla la *cadena* de caracteres descritas por las comillas. Una cadena a veces se conoce como una *cadena de caracteres*, un *mensaje* o una *literal*. Toda la línea, incluyendo a `printf`, sus *argumentos* dentro de los paréntesis, y el *punto y coma* (;), se llama un *enunciado*. Todo enunciado debe terminar con un punto y coma (también conocido como *terminador de enunciado*). Cuando se ejecuta el enunciado anterior `printf`, imprime en pantalla el mensaje `Welcome to C!`. Los caracteres por lo regular se imprimirán exactamente como aparecen entre las dobles comillas del enunciado `printf`. Advierta que los caracteres `\n` no aparecieron impresos en pantalla. La diagonal invertida (\) se llama un *carácter de escape*. Indica que `printf` se supone debe ejecutar algo extraordinario. Cuando se encuentra con una diagonal invertida, `printf` mira hacia adelante, lee el siguiente carácter y lo combina con la diagonal invertida para formar una *secuencia de escape*. La secuencia de escape `\n` significa *nueva línea*, y hace que en pantalla el cursor se coloque al principio de la siguiente línea. Otras secuencias de escape comunes se listan en la figura 2.2. La función `printf` es una de las muchas funciones incluidas en la *Biblioteca estándar de C* (enlistada en el Apéndice B).

| Secuencia de escape | Descripción |
|---|---|
| `\n` | Nueva línea. Coloca el cursor al principio de la siguiente línea. |
| `\t` | Tabulador horizontal. Mueve el cursor al siguiente tabulador. |
| `\r` | Retorno de carro. Coloca el cursor al principio de la línea actual; no avanza a la línea siguiente. |
| `\a` | Alerta. Hace sonar la campana del sistema. |
| `\\` | Diagonal invertida. Imprime un carácter de diagonal invertida en un enunciado `printf`. |
| `\"` | Doble comilla. Imprime un carácter de doble comilla en un enunciado `printf`. |

**Fig. 2.2** Algunas secuencias de escape comunes.

Las dos últimas secuencias de escape de la figura 2.2 pudieran parecer raras. Dada que la diagonal invertida tiene una significación especial para **printf**, es decir, la reconoce como un carácter de escape en vez de un carácter para su impresión, utilizamos una doble diagonal invertida (\\) para indicar que una sola diagonal invertida debe de ser impresa. La impresión de una doble comilla también presenta un problema para **printf**, porque por lo regular supone que una comilla doble marca el límite de una cadena, y que una doble comilla por sí misma, de hecho no debe ser impresa. Al utilizar la secuencia de escape \" le informamos a **printf** que imprima una doble comilla.

La *llave derecha*, }, indica que se ha llegado al final de **main**.

***Error común de programación 2.3***

*Escribir en un programa el nombre de la función de salida **printf** como solo **print**.*

Dijimos que **printf** hace que la computadora ejecute una *acción*. Conforme cualquier programa se ejecuta, lleva a cabo una variedad de acciones y el programa toma *decisiones*. Al final de este capítulo, analizaremos la toma de decisiones. En el capítulo 3, explicaremos con mayor detalle este *modelo de acción/decisión de la programación*.

Es importante advertir que las funciones estándar de biblioteca, como **printf** y **scanf**, no forman parte del lenguaje de programación C. Por lo tanto, por ejemplo, el compilador no podrá encontrar un error de ortografía en **printf** o en **scanf**. Cuando el compilador compila un enunciado **printf**, sólo deja espacio libre en el programa objeto para una "llamada" a la función de biblioteca. Pero el compilador no sabe dónde están las funciones de biblioteca. Quien lo sabe es el enlazador. Por lo tanto, cuando el enlazador se ejecuta, localiza las funciones de biblioteca e inserta las llamadas apropiadas a esas funciones de biblioteca, dentro del programa objeto. Entonces queda el programa objeto "completo" y listo para su ejecución. De hecho, un programa enlazado a menudo se llama un *ejecutable*. Si el nombre de la función está mal escrito, será el enlazador quien encuentre el error, porque no será capaz de hacer coincidir el nombre existente en el programa C con el nombre de cualquier función conocida existente en las bibliotecas.

***Práctica sana de programación 2.2***

*El último carácter impreso por una función que haga cualquier impresión, debería ser una nueva línea (**\n**). Esto asegura que la función dejará el cursor de pantalla colocado al principio de una nueva línea.*



*Prácticas de esta naturaleza fomentan la reutilización del software —una meta clave en los entornos de desarrollo de software.*

***Práctica sana de programación 2.3***

*Haga un nivel de sangría (tres espacios) en todo el cuerpo de cada función dentro de las llaves que definen el cuerpo de la función. Esto enfatiza la estructura funcional de los programas y ayuda a hacer que los programas sean más legibles.*

***Práctica sana de programación 2.4***

*Defina una regla convencional para el tamaño de la sangría que prefiera y a continuación aplíquela de forma uniforme. La tecla del tabulador puede ser utilizada para crear sangrías, pero los tabuladores pudieran variar. Recomendamos utilizar ya sea tabuladores de 1/4 de pulgada, o contar a mano tres espacios por cada uno de los niveles de sangría.*

La función **printf** puede imprimir **Welcome to C!** de varias formas diferentes. Por ejemplo, el programa de la figura 2.3 produce la misma salida o resultado que el programa de la figura 2.1. Esto es así porque **printf** continúa imprimiendo donde se detuvo el anterior **printf** en su impresión. El primer **printf** imprime **Welcome** seguido por un espacio, y el segundo **printf** empieza a imprimir de inmediato, a continuación del espacio.

Un solo **printf** puede imprimir varias líneas, utilizando caracteres de nueva línea como se ve en la figura 2.4. Cada vez que se encuentra con la secuencia de escape **\n** (nueva línea), **printf** se coloca al principio de la siguiente línea.

```
/* Printing on one line with two printf statements */

main()
{
   printf("Welcome ");
   printf("to C!\n");
}
```

```
Welcome to C!
```

**Fig. 2.3** Cómo imprimir en una línea utilizando enunciados separados **printf**.

```
/* Printing multiple lines with a single printf */

main()
{
   printf("Welcome\nto\nC!\n");
}
```

```
Welcome
to
C!
```

**Fig. 2.4** Cómo imprimir en líneas múltiples con un solo **printf**.

## 2.3 Otro programa simple en C: cómo sumar dos enteros

Nuestro siguiente programa utiliza la función **scanf** de la biblioteca estándar para obtener dos enteros escritos sobre el teclado por el usuario, calcular la suma de estos valores, e imprimir el resultado utilizando **printf**. Tanto el programa como la salida de muestra aparecen en la figura 2.5.

El comentario **/* Addition program */** declara el objetivo del programa. La línea

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

es una directriz del *preprocesador de C*. Las líneas que se inician con el signo de # son procesadas por el preprocesador, antes de la compilación del programa. Esta línea de forma específica le indica al preprocesador que incluya dentro del programa el contenido del archivo de *cabecera de entrada/salida estándar* (**stdio.h**). Este archivo de cabecera contiene información y declaraciones utilizadas por el compilador al compilar funciones estándar de biblioteca de entrada y salida como son **printf**. El archivo de cabecera también contiene información que ayuda al compilador a determinar si las llamadas a las funciones de biblioteca han sido escritas de manera correcta. En el capítulo 5 explicaremos con mayor detalle el contenido de los archivos de cabecera.

***Práctica sana de programación 2.5***

*Aunque la inclusión de* **<stdio.h>** *es opcional, deberá ser incluida en cualquier programa C que utilice funciones de entrada/salida estándar de biblioteca. Esto ayuda al compilador a auxiliarle a usted a localizar errores en la fase de compilación de su programa, más bien que en la fase de ejecución (donde los errores resultan más costosos de corregir).*

```
/* Addition program */
#include <stdio.h>

main()
{
   int integer1, integer2, sum;      /* declaration */

   printf("Enter first integer\n");  /* prompt */
   scanf("%d", &integer1);           /* read an integer */
   printf("Enter second integer\n"); /* prompt */
   scanf("%d", &integer2);           /* read an integer */
   sum = integer1 + integer2;        /* assignment of sum */
   printf("Sum is %d\n", sum);       /* print sum */

   return 0;  /* indicate that program ended successfully */
}
```

```
Enter first integer
45
Enter second integer
72
Sum is 117
```

Fig. 2.5 Un programa de suma.



Como se ha indicado, todos los programas inician su ejecución con **main**. La llave izquierda { marca el principio del cuerpo de **main** y la llave derecha correspondiente marca el final de **main**. La línea

```
int integer1, integer2, sum;
```

es una *declaración*. Las letras **integer1**, **integer2**, y **sum** son los nombres de *variables*. Una variable es una posición en memoria donde se puede almacenar un valor para uso de un programa. Esta declaración especifica que las variables **integer1**, **integer2**, y **sum** son del tipo ***int***, lo que significa que estas variables contendrán valores *enteros*, es decir, valores tales como 7, -11, 0, 31914, y similares. Todas las variables deben de declararse con un nombre y un tipo de datos, de inmediato después de la llave izquierda que inicia el cuerpo de **main**, antes de que puedan ser utilizadas en un programa. En C existen otros tipos de datos, además de **int**. En una declaración se pueden declarar varias variables del mismo tipo. Podíamos haber escrito tres declaraciones, una para cada variable, pero la declaración anterior es más concisa.

***Práctica sana de programación 2.6***

*Coloque un espacio después de cada coma (,) para hacer los programas más legibles.*

Un nombre de variable en C es cualquier *identificador* válido. Un identificador es una serie de caracteres formados de letras, dígitos y subrayados ( _ ) que no se inicien con un dígito. Un identificador puede tener cualquier longitud, pero según con el estándar ANSI C sólo se requieren los primeros 31 caracteres para su reconocimiento por los compiladores de C. C es *sensible* —lo que quiere decir que para C las letras mayúsculas y minúsculas son diferentes, por lo que **a1** y **A1** son identificadores distintos.

***Error común de programación 2.4***

*Usar una letra mayúscula donde debería haberse usado una minúscula (por ejemplo al escribir* ***Main*** *en vez de* ***main****).*

***Sugerencia de portabilidad 2.1***

*Utilice identificadores de 31 o menos caracteres. Esto auxilia a asegurar la portabilidad y puede evitar algunos errores sutiles de programación.*

***Práctica sana de programación 2.7***

*La selección de nombres de variables significativos ayuda a la autodocumentación de un programa, es decir, se requerirán de menos comentarios.*

***Práctica sana de programación 2.8***

*La primera letra de un identificador utilizado como un nombre simple de variable, deberá ser una letra minúscula. Más adelante, en el texto, le daremos significación especial a aquellos identificadores que empiezan con una letra mayúscula y aquellos que utilizan todas mayúsculas.*

***Práctica sana de programación 2.9***

*Los nombres de variables de varias palabras pueden auxiliar a la legibilidad de un programa. Evite juntar palabras separadas, como en* `totalcommissions`*. En vez de ello, separe las palabras con subrayados como* `total_commissions`*, o bien, si desea reunirlas, empiece cada palabra después de la primera con una letra mayúscula, como en* `totalCommissions`*.*

Las declaraciones deben de ser colocadas después de la llave izquierda de una función y antes de *cualquier* enunciado ejecutable. Por ejemplo, en el programa de la figura 2.5, la inserción de la declaración después de la primera **printf** generaría un error de sintaxis. Se causa un *error de sintaxis* cuando el compilador no puede reconocer un enunciado. El compilador por lo general emite un mensaje de error para auxiliar al programador en la localización y corrección del enunciado incorrecto. Los errores de sintaxis son violaciones al lenguaje. Los errores de sintaxis también se conocen como *errores de compilación o errores en tiempo de compilación.*

***Error común de programación 2.5***

*Colocar declaraciones de variables entre enunciados ejecutables.*

***Práctica sana de programación 2.10***

*Separe las declaraciones y los enunciados ejecutables en una función mediante una línea en blanco, para enfatizar dónde terminan las declaraciones y dónde empiezan los enunciados ejecutables.*

El enunciado

```
printf ("Enter first integer\n");
```

imprime la literal **Enter first integer** en la pantalla y se coloca al principio de la línea siguiente. Este mensaje se llama un porque le dice al usuario que debe de tomar una acción específica.

El enunciado

```
scanf ("%d", &integer1);
```

utiliza ***scanf*** para obtener un valor del usuario. La función **scanf** toma la entrada de la entrada estándar, que normalmente es el teclado. Este **scanf** tiene dos argumentos, **"%d"** y **&integer1**. El primer argumento, la *cadena de control de formato*, indica el tipo de dato que deberá ser escrito por el usuario. El *especificador de conversión* **%d** indica que los datos deberán de ser un entero (la letra **d** significa "entero decimal"). El signo de % en este contexto es tratado por **scanf** (y como veremos por **printf**) como un carácter de escape (como \)y la combinación **%d** es una secuencia de escape (como sería **\n**). El segundo argumento de **scanf** empieza con un ampersand (**&**) —llamado el *operador de dirección* en C— seguido por un nombre de variable. El ampersand, al ser combinado con el nombre de variable, le indica a **scanf** la posición en memoria en la cual está almacenada la variable **integer1**. La computadora a continuación almacena el valor correspondiente a **integer1** en dicha posición. El uso de ampersand (**&**) resulta con frecuencia confuso para los programadores neófitos o para personas que hayan programado en otros lenguajes que no requieren de esta notación. Por ahora, sólo recuerde de anteceder cada variable en todos los enunciados **scanf** con un ampersand. Algunas excepciones a esta regla, se analizan en los capítulos 6 y 7. El verdadero significado del uso del ampersand será aclarado una vez que estudiemos los apuntadores en el capítulo 7.

Cuando la computadora ejecute el **scanf** anterior, esperará a que el usuario escriba un valor para la variable **integer1**. El usuario responderá escribiendo un entero y presionando la *tecla de retorno* (a veces conocida como *tecla de entrar*) para enviar el número a la computadora. La computadora entonces asigna este número, o *valor* a la variable **integer1**. Cualquier referencia subsecuente a **integer1** dentro del programa utilizará este mismo valor. Las funciones **printf** y **scanf** facilitan la interacción entre el usuario y la computadora. Dado que esta interacción se



asemeja a un diálogo, a menudo se conoce como *computación conversacional* o *computación interactiva.*

El enunciado

```
printf ("Enter second integer\n");
```

imprime el mensaje **Enter second integer** en la pantalla, y a continuación posiciona el cursor al principio de la siguiente línea. También este **printf** indica al usuario que tome acción.

El enunciado

```
scanf ("%d", &integer2);
```

obtiene un valor para la variable **integer2.** El *enunciado de asignación*

```
sum = integer1 + integer2;
```

calcula la suma de las variables **integer1** e **integer2** y asigna el resultado a la variable **sum** utilizando el *operador de asignación* **=**. El enunciado se lee como, "**sum** *obtiene* el valor de **integer1 + integer2**". La mayor parte de los cálculos se ejecutan en enunciados de asignación. El operador = y el operador + se llaman *operadores binarios,* porque cada uno de ellos tiene *dos operandos.* En el caso del operador **+**, los dos operandos son **integer1** e **integer2**. En el caso del operador **=**, los dos operandos son **sum** y el valor de la expresión **integer1 + integer2**.

***Práctica sana de programación 2.11***

*Coloque espacios en blanco a ambos lados de un operador binario. Esto hace resaltar al operador y hace que el programa sea mas legible.*

***Error común de programación 2.6***

*El cálculo en un enunciado de asignación debe de aparecer en el lado derecho del operador =. Es un error de sintaxis colocar un cálculo del lado izquierdo de un operador de asignación.*

El enunciado

```
printf("Sum is %d\n", sum);
```

utiliza la función **printf** para imprimir en pantalla la literal **Sum is** seguida del valor numérico de la variable **sum**. Esta **printf** tiene dos argumentos, **"Sum is %d\n"** y **sum**. El primer argumento es la cadena de control de formato. Contiene algunos caracteres literales que se desplegarán, así como el especificador de conversión **%d**, que indica que deberá imprimirse un entero. El segundo argumento especifica el valor a ser impreso. Advierta que el especificador de conversión para un entero es idéntico tanto en **printf** como en **scanf**. Esto es cierto para la mayor parte de los tipos de datos en C.

Los cálculos también pueden llevarse a cabo dentro de enunciados **printf**. Podríamos haber combinado los dos enunciados previos en el siguiente enunciado

```
printf("Sum is %d\n", integer1 + integer2);
```

El enunciado

```
return 0;
```

pasa el valor 0 de regreso al entorno del sistema operativo en el cual se está ejecutando el programa. Esto le indica al sistema operativo que el programa fue ejecutado con éxito. Para más información sobre cómo informar de algún tipo de falla del programa, refiérase a los manuales de su entorno de sistema operativo particular.

La llave derecha, }, indica que se ha llegado al final de la función **main**.

***Error común de programación 2.7***

*Olvidarse uno o ambos de las dobles comillas que rodean a la cadena de control de formato en un* `printf` *o un* `scanf`.

***Error común de programación 2.8***

*Olvidar el signo de* `%` *en una especificación de conversión en la cadena de control de formato de un* `printf` *o* `scanf`.

***Error común de programación 2.9***

*Colocar una secuencia de escape como* `\n` *fuera de la cadena de control de formato de un* `printf` *o de un* `scanf`.

***Error común de programación 2.10***

*Olvidar incluir las expresiones cuyos valores deben de ser impresas en una* `printf` *que contiene especificadores de conversión.*

***Error común de programación 2.11***

*No proporcionar en una cadena de control de formato* `printf` *un especificador de conversión, cuando se requiere de uno para imprimir una expresión.*

***Error común de programación 2.12***

*Colocar dentro de la cadena de control de formato la coma, que se supone debe separar la cadena de control de formato de la expresiones a imprimirse.*

***Error común de programación 2.13***

*Olvidar anteceder una variable en un enunciado* `scanf` *con un ampersand, cuando dicha variable debería, de hecho, estar precedida por un ampersand.*

En muchos sistemas, este error de tiempo de ejecución se conoce como una "falla de segmentación" o bien una "violación de acceso". Este tipo de error ocurre cuando el programa de un usuario intenta tener acceso a una parte de la memoria de la computadora, en la cual el programa del usuario no tiene privilegios de acceso. La causa precisa de este error será explicada en el capítulo 7.

***Error común de programación 2.14***

*Anteceder una variable incluida en un enunciado* `printf` *con un ampersand, cuando de hecho esa variable no debería ser precedida por un ampersand.*



En el capítulo 7, estudiaremos apuntadores y veremos casos en los cuales desearemos anteceder un nombre de variable con un ampersand para imprimir la dirección de dicha variable. Durante los siguientes varios capítulos, sin embargo, los enunciados **printf** no deberán de incluir estos ampersand.

## 2.4 Conceptos de memoria

Los nombres de variables como **integer1**, **integer2** y **sum** de hecho corresponden a *localizaciones* o *posiciones* en la memoria de la computadora. Cada variable tiene un *nombre*, un *tipo* y un *valor*.

En el programa de suma de la figura 2.5, cuando se ejecuta el enunciado

```
scanf ("%d", &integer1);
```

el valor escrito por el usuario se coloca en una posición de memoria a la cual se ha asignado el nombre **integer1**. Suponga que el usuario escribe el número **45** como el valor para **integer1**. La computadora colocará **45** en la posición **integer1**, como se muestra en la figura 2.6.

Siempre que se coloca un valor en una posición de memoria, este valor sustituye el valor anterior existente en dicha posición. Dado que la información previa resulta destruida, el proceso de leer información a la memoria se llama *lectura destructiva*.

Regresando de nuevo a nuestro programa de suma, cuando el enunciado

```
scanf ("%d", &integer2);
```

se ejecuta, suponga que el usuario escribe el valor **72**. Este valor se coloca en la posición **integer2**, y la memoria aparece como se muestra en la figura 2.7. Advierta que en memoria estas posiciones no necesariamente serán adyacentes.

Una vez que el programa ha obtenido los valores de **integer1** e **integer2**, suma estos valores y coloca la suma en la variable **sum**. El enunciado

```
sum = integer1 + integer2;
```


Fig. 2.6 Una posición de memoria mostrando el nombre y valor de una variable.

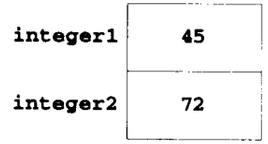



Fig. 2.7 Posiciones de memoria una vez que se han introducido ambas variables.

que ejecuta la suma también involucra lectura destructiva. Esto ocurre cuando la suma calculada de **integer1** y de **integer2** se coloca en la posición **sum** (destruyendo el valor que pudiera haber existido ya en **sum**). Una vez que **sum** esté calculada, la memoria aparece como en la figura 2.8. Note que los valores de **integer1** e **integer2** aparecen de forma exacta como estaban antes de haber sido utilizados en el cálculo de **sum**. Estos valores fueron utilizados pero no destruidos, al ejecutar el cálculo a la computadora. Entonces, cuando un valor se lee de la posición de memoria, el proceso se conoce como *lectura no destructiva.*

## 2.5 Aritmética en C

La mayor parte de los programas C ejecutan cálculos aritméticos. Los *operadores aritméticos de C* se resumen en la figura 2.9. Advierta el uso de varios símbolos especiales, no utilizados en álgebra. El *asterisco* (*) indica multiplicación y el *signo de por ciento* (%) denota el operador *módulo,* que se presenta más adelante. En álgebra, si deseamos multiplicar *a* por *b* simplemente colocamos estos nombres de variable de una sola letra uno al lado del otro, como en *ab.* En C, sin embargo, si hiciéramos esto, **ab** se interpretaría como un nombre solo de dos letras (o un identificador). Por lo tanto, C (y en general otros lenguajes de programación) requiere que se denote la multiplicación en forma explícita, utilizando el operador *****, como en **a*b**.

Los operadores aritméticos son todos operadores binarios. Por ejemplo, la expresión **3 + 7** contiene el operador binario + y los operandos **3** y **7**.

| | |
|---|---|
| **integer1** | **45** |
| **integer2** | **72** |
| **sum** | **117** |

**Fig. 2.8 Localizaciones de memoria después de un cálculo.**

| Operación en C | Operador aritmético | Expresión algebraica | Expresión en C |
|---|---|---|---|
| Suma | + | f + 7 | **f + 7** |
| Substracción | - | p - c | **p - c** |
| Multiplicación | * | bm | **b * m** |
| División | / | x / y o $\frac{x}{y}$ o $x \div y$ | **x / y** |
| Módulo | % | r mod s | **r % s** |

**Fig. 2.9** Operadores aritméticos de C.



La *división de enteros* da como resultado también un entero. Por ejemplo, la expresión **7/4** da como resultado **1**, y la expresión **17/5** da como resultado **3**. C tiene el operador de módulo, %, que proporciona el residuo después de una división de enteros. El operador de módulo es un operador entero, que puede ser utilizado sólo con operandos enteros. La expresión **x%y** resulta o entrega el residuo, después de que **x** haya sido dividido por **y**. Por lo tanto, **7%4** da como resultado **3**, y **17%5** da como resultado **2**. Analizarémos muchas aplicaciones interesantes del operador de módulo.

***Error común de programación 2.15***

*Un intento de dividir entre cero, por lo regular resulta no definido en sistemas de cómputo y en general da como resultado un error fatal, es decir, un error que hace que el programa se termine de inmediato sin haber ejecutado con éxito su tarea. Los errores no fatales permiten que los programas se ejecuten hasta su término, produciendo a menudo resultados incorrectos.*

Las expresiones aritméticas en C deben de ser escritas en una *línea continua,* para facilitar la escritura de programas en la computadora. Entonces, expresiones tales como "**a** dividido por **b**" debe de estar escrito como **a/b,** de tal forma que todos los operadores y operandos aparezcan en una sola línea. La notación algebraica

$$\frac{a}{b}$$

en general no es aceptable para compiladores, aunque si existen algunos paquetes de software de uso especial que aceptan notaciones más naturales, para expresiones matemáticas complejas.

Los paréntesis se utilizan en las expresiones de C de manera muy similar a como se usan en las expresiones algebraicas. Por ejemplo, para multiplicar **a** por la cantidad **b+c** escribimos:

```
a * (b + c)
```

C calculará las expresiones aritméticas en una secuencia precisa, determinada por las *reglas de precedencia de operadores* que siguen y, que en general son las mismas que las que se siguen en álgebra.

1. Primero se calculan expresiones o porciones de expresiones contenidas dentro de pares de paréntesis. Entonces, *los paréntesis pueden ser utilizados para obligar a un orden de evaluación en cualquier secuencia deseada por el evaluador. Los paréntesis* se dicen que están en el "más alto nivel de precedencia". En el caso de paréntesis *anidados* o *incrustados,* se evalúa primero la expresión en el par de paréntesis más interno.
2. A continuación, se calculan las operaciones de multiplicación, división y módulo. Si una expresión contiene varias multiplicaciones, divisiones y módulos, la evaluación avanzará de izquierda a derecha. Se dice que la multiplicación, división y módulo tienen el mismo nivel de precedencia.
3. Por último, se calculan las operaciones de suma y de resta. Si una expresión contiene varias operaciones de suma y de resta, la evaluación avanzará de izquierda a derecha. La suma y la resta también tienen el mismo nivel de precedencia.

Las reglas de precedencia de operadores son guías de acción, que le permiten a C calcular expresiones en el orden correcto. Cuando decimos que una evaluación o cálculo avanza de izquierda a derecha, nos estamos refiriendo a la *asociatividad* de los operadores. Veremos que algunos operadores se asocian de derecha a izquierda. En la figura 2.10 se resumen estas reglas de precedencia de operadores.

| Operador (es) | Operación (es) | Orden de cálculo (precedencia) |
|---|---|---|
| ( ) | Paréntesis | Se calculan primero. Si los paréntesis están anidados, la expresión en el par más interno se evalúa primero. Si existen varios pares de paréntesis "en el mismo nivel" (es decir no anidadas), se calcularán de izquierda a derecha. |
| *, /, o bien % | Multiplicación, División y Módulo | Se evalúan en segundo lugar. Si existen varias, se calcularán de izquierda a derecha. |
| + o bien - | Suma o Resta | Se calculan al último. Si existen varios, serán evaluados de izquierda a derecha. |

**Fig. 2.10** Precedencia de operadores aritméticos.

Veamos ahora varias expresiones a la luz de las reglas de precedencia de operadores. Cada ejemplo enlista una expresión algebraica y su equivalente en C.

El ejemplo siguiente calcula el promedio aritmético (la media) de cinco términos:

Algebra: $m = \frac{a + b + c + d + e}{5}$

C:
```
m = (a + b + c + d + e) / 5;
```

Se requieren los paréntesis, porque la división tiene una precedencia más alta que la suma. Toda la cantidad `(a + b + c + d + e)` debe de dividirse entre `5`. Si los paréntesis se omiten por error, `a + b + c + d + e / 5` que se calcula incorrectamente como

$$a + b + c + d + \frac{e}{5}$$

El siguiente ejemplo es la ecuación de una línea recta:

Algebra: $y = mx + b$

C:
```
y = m * x + b;
```

No se requieren paréntesis. Primero se evalúa la multiplicación, porque ésta tiene una precedencia más alta que la suma.

El siguiente ejemplo contiene módulo (%), multiplicación, división, adición y substracción:

Algebra: $z = pr\%q + w/x - y$

C:
```
z =   p * r % q + w / x - y;
      1   2   4   3   5
```



Los números en círculos bajo el enunciado indican el orden en el cual valorará o calculará C los operadores. La multiplicación, el módulo y la división, serán evaluadas primero, en un orden de izquierda a derecha (es decir, se asocian de izquierda a derecha), en vista de que tienen precedencia mayor que la suma y la resta. La suma y la resta serán calculadas después. También ellas se evaluarán de izquierda a derecha.

No todas las expresiones con varios pares de paréntesis contienen paréntesis anidados. La expresión

```
a * (b + c) + c * (d + e)
```

no contiene paréntesis anidados. En vez de ello, se dice que estos paréntesis están "en el mismo nivel". En esta situación, se calcularán las expresiones C en paréntesis primero y en un orden de izquierda a derecha.

Para desarrollar una mejor comprensión de las reglas de precedencia de operadores, veamos cómo se calcula un polinomio de segundo grado.

```
y =   a * x * x + b * x + c;
      1   2   4   3   5
```

Los números en círculos bajo el enunciado indican el orden en el cual ejecuta C las operaciones. En C no existe un operador aritmético para exponenciación, por lo cual hemos representado $x^2$ como `x * x`. La biblioteca C estándar incluye la función `pow` ("potencia") para ejecutar la exponenciación. Debido a ciertos aspectos sutiles relacionados con los tipos de datos requeridos por `pow`, vamos a posponer la explicación detallada de `pow` hasta el capítulo 4.

Suponga `a = 2, b =3, c=7,` y `x = 5`. En la figura 2.11 se ilustra cómo se calcula el polinomio de segundo grado ya mostrado.

## 2.6 Toma de decisiones: Operadores de igualdad y relacionales

Los enunciados ejecutables de C, llevan a cabo ya sea *acciones* (como son cálculos o entradas o salidas de datos), o toman *decisiones* (veremos pronto varios ejemplos de éstos). Pudiéramos tomar una decisión en un programa, por ejemplo, para determinar si la calificación de una persona en un examen es mayor que o igual a 60, y si así es, imprimir el mensaje "¡Felicitaciones! usted pasó". Esta sección introduce una versión simple de la *estructura de control* `if` de C, que permite a un programa tomar una decisión basada en la veracidad o falsedad de alguna declaración de hecho, conocida como una *condición*. Si la condición se cumple (si la condición es *cierta o verdadera*) se ejecuta el enunciado existente en el cuerpo de la estructura `if`. Si la condición no se cumple (es decir, si la condición es *falsa*) el enunciado del cuerpo no se ejecuta. Independientemente de que se ejecute o no el enunciado del cuerpo, una vez terminada la estructura `if`, la ejecución sigue adelante con el enunciado, después de la estructura `if`.

Las condiciones en las estructuras `if` se forman utilizando los *operadores de igualdad* y los *operadores relacionales* resumidos en la figura 2.12. Los operadores relacionales tienen un mismo nivel de precedencia y se asocian de izquierda a derecha. Los operadores de igualdad tienen un nivel de precedencia menor que los operadores relacionales y también se asocian de izquierda a derecha. (Nota: en C, una condición puede de hecho ser cualquier expresión que genere un cero (falsa) o un no cero (verdadera). Veremos muchas aplicaciones de lo anterior a lo largo de este libro).

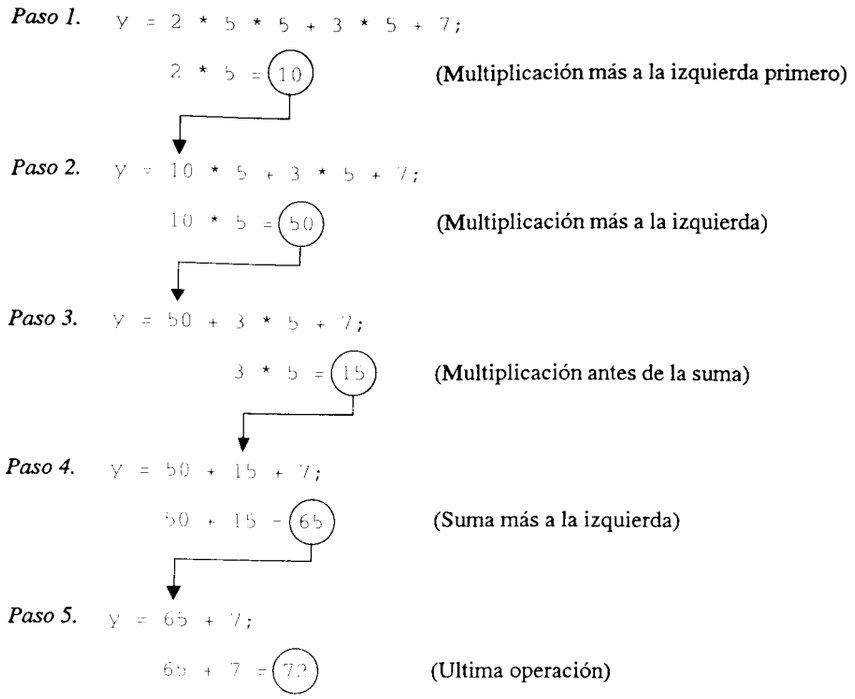



**Fig. 2.11** Cálculo y un polinomio de segundo grado.

| Operador de igualdad estándar algebraico u operador relacional | Operador de igualdad o relacional de C | Ejemplo de condición de C | Significado de la condición de C |
|---|---|---|---|
| Operadores de igualdad | | | |
| = | == | `x == y` | `x` es igual a `y` |
| ≠ | != | `x != y` | `x` no es igual a `y` |
| Operadores relacionales | | | |
| > | > | `x > y` | `x` es mayor que `y` |
| < | < | `x < y` | `x` es menor que `y` |
| ≥ | >= | `x >= y` | `x` es mayor que o igual a `y` |
| ≤ | <= | `x <= y` | `x` es menor que o igual a `y` |

**Fig. 2.12** Operadores de igualdad y relacionales.



***Error común de programación 2.16***

*Ocurrirá un error de sintaxis si los dos símbolos en cualquiera de los operadores `==`, `!=`, `>=` y `<=` están separados por espacios.*

***Error común de programación 2.17***

*Ocurrirá un error de sintaxis si los dos símbolos en cualquiera de los operadores `!=`, `>=`, y `<=`, se invierten como en `=!`, `=>`, y `=<`, respectivamente.*

***Error común de programación 2.18***

*Confundir el operador de igualdad `==` con el operador de asignación `=`.*

Para evitar esta confusión, el operador de igualdad deberá ser leído como "doble igual" y el operador de asignación debe de ser leído como "obtiene". Como veremos pronto, confundir estos dos operadores no podría causar por necesidad un error de sintaxis fácil de reconocer, pero pudiera causar errores lógicos muy sutiles.

***Error común de programación 2.19***

*Colocar un punto y coma inmediatamente a la derecha del paréntesis derecho, después de la condición, en una estructura `if`.*

El ejemplo de la figura 2.13 utiliza seis enunciados `if` para comparar dos números escritos por el usuario. Si se satisface la condición en cualquiera de estos enunciados `if`, se ejecuta el enunciado `printf` asociado con dicho `if`. El programa así como tres salidas de ejecución de muestra aparecen en la figura.

Advierta que el programa de la figura 2.13 utiliza `scanf` para recibir la entrada de dos números. Cada especificador de conversión tiene un argumento correspondiente, en el cual se almacenará un valor. El primer `%d` convierte un valor a almacenar en la variable `num1` y el segundo `%d` convierte un valor a almacenar en la variable `num2`. El hacer una sangría en el cuerpo de cada enunciado `if` y colocar líneas en blanco por arriba y por abajo de cada enunciado `if` mejora la legibilidad del programa. También, note que cada enunciado `if` de la figura 2.13 tiene en su cuerpo un solo enunciado. En el capítulo 3 mostrarémos cómo especificar enunciados `if` con cuerpos de varios enunciados.

***Práctica sana de programación 2.12***

*Haga una sangría en el o los enunciados del cuerpo de una estructura `if`.*

***Práctica sana de programación 2.13***

*Coloque una línea en blanco antes y después de cada estructura de control en un programa para mejor legibilidad.*

***Práctica sana de programación 2.14***

*Deberá procurar que no exista más de un enunciado en cada línea de un programa.*

***Error común de programación 2.20***

*Colocar comas (cuando no se necesita ninguna) entre los especificadores de conversión en la cadena de control de formato en un enunciado `scanf`.*

```
/* Using if statements, relational
   operators, and equality operators */
#include <stdio.h>

main()
{
   int num1, num2;

   printf("Enter two integers, and I will tell you\n");
   printf("the relationships they satisfy: ");
   scanf("%d%d", &num1, &num2);   /* read two integers */

   if (num1 == num2)
      printf("%d is equal to %d\n", num1, num2);

   if (num1 != num2)
      printf("%d is not equal to %d\n", num1, num2);

   if (num1 < num2)
      printf("%d is less than %d\n", num1, num2);

   if (num1 > num2)
      printf("%d is greater than %d\n", num1, num2);

   if (num1 <= num2)
      printf("%d is less than or equal to %d\n",
             num1, num2);

   if (num1 >= num2)
      printf("%d is greater than or equal to %d\n",
             num1, num2);

   return 0;   /* indicate program ended successfully */
}
```

**Fig. 2.13** Cómo utilizar operadores de igualdad y relacionales (parte 1 de 2).

El comentario en la figura 2.13 está dividido sobre dos líneas o renglones. En los programas C, los caracteres de *espacio en blanco* como tabuladores, nueva línea, y espacios, por lo regular son ignorados. Por lo tanto, los enunciados y comentarios pueden ser divididos entre varios renglones. No es correcto, sin embargo, dividir identificadores.

***Práctica sana de programación 2.15***

*Un enunciado largo puede ser dividido en varios renglones. Si un enunciado debe ser dividido en varias líneas, escoja puntos de corte que tengan sentido (como sería después de una coma en una lista separada por comas). Si un enunciado se divide en dos o más renglones, haga una sangría en todas las líneas subsecuentes.*

La gráfica de la figura 2.14 muestra la precedencia de los operadores presentados en este capítulo. Los operadores se muestran de arriba hacia abajo en orden decreciente de precedencia. Advierta que el signo igual es también un operador. Todos estos operadores, a excepción del operador de asignación =, se asocian de izquierda a derecha. El operador de asignación (=)se asocia de derecha a izquierda.



```
Enter two integer, and I will tell you
the relationships they satisfy: 3 7
3 is not equal to 7
3 is less than 7
3 is less than or equal to 7
```

```
Enter two integers, and I will tell you
the relationships they satisfy: 22 12
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12
```

```
Enter two integers, and I will tell you
the relationships they satisfy: 7 7
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7
```

**Fig. 2.13** Cómo utilizar operadores de igualdad y relacionales (parte 2 de 2).

***Practica sana de programación 2.16***

*Refiérase a la gráfica de precedencia de operadores al escribir expresiones que contengan muchos operadores. Confirme que los operadores en la expresión se ejecutan en el orden correcto. Si no está totalmente seguro del orden de evaluación en una expresión compleja, utilice paréntesis para obligar a un orden, exactamente como lo haría en expresiones algebraicas. Asegúrese de observar que algunos de los operadores de C, como el operador de asignación (=) se asocian de derecha a izquierda, en vez de izquierda a derecha.*

| Operadores | Asociatividad |
|---|---|
| ( ) | de izquierda a derecha |
| * / % | de izquierda a derecha |
| + - | de izquierda a derecha |
| < <= > >= | de izquierda a derecha |
| == != | de izquierda a derecha |
| = | de derecha a izquierda |

**Fig. 2.14** Precedencia y asociatividad de los operadores hasta ahora analizados.

Algunas de las palabras que hemos utilizado en los programas de C de este capítulo en particular **int**, **return** e **if** son palabras reservadas del lenguaje. El conjunto completo de palabras reservadas de C se muestra en la figura 2.15. Estas palabras tienen un significado especial para el compilador de C, por lo que el programador debe tener cuidado de no utilizar estas palabras como identificadores como serían nombres de variables. En este libro, analizaremos todas estas palabras reservadas.

En este capítulo, hemos presentado muchas características importantes del lenguaje de programación de C, incluyendo la impresión de datos en la pantalla, introducir datos del usuario, llevar a cabo cálculos y tomar decisiones. En el siguiente capítulo elaboraremos más sobre estas técnicas, conforme nos iniciamos en la *programación estructurada*. El estudiante se irá familiarizando con las técnicas de sangría. Estudiaremos cómo especificar el orden en el cual se ejecutan los enunciados —esto se llama *flujo de control*.

## Resumen

- Los comentarios empiezan con **/*** y terminan con ***/**. Los programadores insertan comentarios para documentar los programas y mejorar su legibilidad. Los comentarios no hacen que la computadora lleve a cabo acción alguna cuando se ejecute el programa.
- La directriz de preprocesador #include **<stdio.h>** le indica al compilador que incluya el archivo de cabecera de entrada/salida estándar dentro del programa. Este archivo contiene información utilizada por el compilador para verificar la precisión de las llamadas de funciones de entrada y de salida, como son **scanf** y **printf**.
- Los programas de C están formados de funciones, una de las cuales debe de ser **main**. Todos los programas de C empiezan su ejecución en la función **main**.
- La función **printf** puede ser utilizada para imprimir una cadena contenida entre comillas, y para imprimir los valores de expresiones. Al imprimir valores enteros, el primer argumento de una función **printf** —la cadena de control de formato— contiene el especificador de conversión **%d** y cualquier otros caracteres que deban de ser impresos; el segundo argumento es la expresión cuyo valor será impresa. Si más de un entero se imprimirá, entonces la cadena de control de formato contendrá un **%d** para cada uno de los enteros, y los argumentos separados con coma, que siguen a la cadena de control de formato, contendrán las expresiones cuyos valores serán o irán a ser impresos.

**Palabras clave**

| | | | |
|---|---|---|---|
| **auto** | **break** | **case** | **char** |
| **const** | **continue** | **default** | **do** |
| **double** | **else** | **enum** | **extern** |
| **float** | **for** | **goto** | **if** |
| **int** | **long** | **register** | **return** |
| **short** | **signed** | **sizeof** | **static** |
| **struct** | **switch** | **typedef** | **union** |
| **unsigned** | **void** | **volatile** | **while** |

**Fig. 2.15** Palabras reservadas de C.



- La función **scanf** obtiene valores que el usuario por lo regular escribe en el teclado. Su primer argumento es la cadena de control de formato, que le indica a la computadora cuál es el tipo de datos que debe ser introducido por el usuario. El especificador de conversión %d indica que los datos deberán de ser un entero. Cada uno de los argumentos restantes corresponde a uno de los especificadores de conversión existentes en la cadena de control de formato. Cada nombre de variable normalmente es precedida por un ampersand (**&**), conocido como el operador de dirección en C. El ampersand, al ser combinado con un nombre de variable, le indica a la computadora la posición en memoria donde se almacenará el valor. La computadora entonces almacena el valor en esa posición.
- Todas las variables en un programa C deben ser declaradas antes de que puedan ser utilizadas por el programa.
- Un nombre de variable en C es cualquier identificador válido. Un identificador es una serie de caracteres consistiendo de letras, dígitos y subrayados (_). Los identificadores no pueden empezar con un dígito. Los identificadores pueden tener cualquier longitud; sin embargo, sólo los primeros 31 caracteres son significativos, de acuerdo con el estándar ANSI.
- C diferencia minúsculas de mayúsculas.
- La mayor parte de los cálculos se ejecutan en enunciados de asignación.
- Todas las variables almacenadas en la memoria de la computadora tienen un nombre, un valor y un tipo.
- Siempre que se coloque un nuevo valor en una posición en memoria, substituye el valor anterior existente en esa misma posición. Dado que esta información anterior queda destruida, el proceso de leer información hacia la memoria se conoce como lectura destructiva.
- El proceso de leer un valor desde la memoria se conoce como lectura no destructiva.
- Las expresiones aritméticas en C deben ser escritas en un solo renglón, para facilitar escribir programas en la computadora.
- C calcula las expresiones aritméticas en una secuencia precisa, definida por las reglas de precedencia de operadores y de asociatividad.
- El enunciado **if** permite al programador tomar una decisión cuando se cumple cierta condición. El formato para un enunciado **if** es

  **if** (*condición*)
  *enunciado*

- Si la condición es verdadera, el enunciado en el cuerpo de **if** se ejecuta. Si la condición es falsa, el enunciado del cuerpo será saltado.
- Las condiciones en los enunciados **if**, se forman por lo común mediante el uso de operadores de igualdad y operadores relacionales. El resultado de utilizar estos operadores es siempre la observación de "verdadero" o "falso". Note que las condiciones pueden ser cualquier expresión que genere un valor cero (falso) o un valor no cero (verdadero).

## Terminología

acción
modelo de acción/decisión
operador de dirección
ampersand (**&**)
argumento
operadores aritméticos

operador de asignación (=)
enunciado de asignación
asociatividad de operadores
asterisco (*)
carácter de escape diagonal invertida (\)
operadores binarios
bloque
cuerpo de una función
llaves {}
C
diferencia minúsculas de mayúsculas
cadena de caracteres
palabras reservada de C
comentario
error de compilación
error en tiempo de compilación
condición
cadena de control
computación conversacional
especificador de conversión
preprocesador de C
Biblioteca estándar de C
especificador de conversión %d
decisión
toma de decisiones
declaración
lectura destructiva
división entre cero
tecla de entrar
operador de asignación signo igual (=)
operadores de igualdad
  == "igual a"
  != "no es igual a"
carácter de escape
secuencia de escape
falso
error fatal
flujo de control
cadena de control de formato
función
identificador
estructura de control **if**
sangría
**int**
entero
división de enteros
computación interactiva
palabras clave
asociatividad de izquierda a derecha
literal
posición
**main**
memoria
posición en memoria
mensaje
operador de módulo (%)
operador de multiplicación (*)
**nombre**
paréntesis anidados
carácter de nueva línea (\n)
lectura no destructiva
error no fatal
no cero (verdadero)
operando
operador
paréntesis ()
carácter de escape signo de por ciento (%)
precedencia
función **printf**
indicador
operadores relacionales
  > "es mayor que"
  < "es menor que"
  >= "es mayor que o igual a"
  <= "es menor que o igual a"
palabras reservadas
tecla de retorno
asociatividad de derecha a izquierda
reglas de precedencia de operadores
función **scanf**
terminador de enunciado punto y coma (;)
archivo de cabecera de entrada/salida estándar
enunciado
terminador de enunciado (;)
**stdio.h**
en línea recta
cadena
programación estructurada
error de sintaxis
verdadero
subrayado ( _ )
valor
variable
nombre de variable
tipo de variable
valor de variable
caracteres de espacio en blanco
cero (falso)



## *Errores comunes de programación*

**2.1** Olvidar terminar un comentario con */
**2.2** Iniciar un comentario con los caracteres */ o terminar un comentario con los caracteres /*
**2.3** Escribir en un programa el nombre de la función de salida **printf** como solo **print.**
**2.4** Usar una letra mayúscula donde debería haberse usado una minúscula (por ejemplo al escribir **Main** en vez de **main**).
**2.5** Colocar declaraciones de variables entre enunciados ejecutables.
**2.6** El cálculo en un enunciado de asignación debe de aparecer en el lado derecho del operador =. Es un error de sintaxis colocar un cálculo del lado izquierdo de un operador de asignación.
**2.7** Olvidarse uno o ambas de las dobles comillas que rodean a la cadena de control de formato en un **printf** o un **scanf**.
**2.8** Olvidar el signo de % en una especificación de conversión en la cadena de control de formato de un **printf** o **scanf**.
**2.9** Colocar una secuencia de escape como \n fuera de la cadena de control de formato de un **printf** o de un **scanf**.
**2.10** Olvidar incluir las expresiones cuyos valores deben de ser impresas en un **printf** que contiene especificadores de conversión.
**2.11** No proporcionar en una cadena de control de formato **printf** un especificador de conversión cuando se requiere de uno para imprimir una expresión.
**2.12** Colocar dentro de la cadena de control de formato la coma que se supone debe separar la cadena de control de formato de las expresiones a imprimirse.
**2.13** Olvidar de anteceder una variable en un enunciado **scanf** con un ampersand cuando dicha variable debería, de hecho, estar precedida por un ampersand.
**2.14** Anteceder una variable incluida en un enunciado **printf** con un ampersand, cuando, de hecho esa variable no debería ser precedida por un ampersand.
**2.15** Un intento de dividir entre cero, por lo regular resulta no definido en sistemas de cómputo y en general da como resultado fatal, es decir, un error que hace que el programa se termine de inmediato sin haber ejecutado de forma exitosa su tarea. Los errores no fatales permiten que los programas se ejecuten hasta su término, produciendo a menudo resultados incorrectos.
**2.16** Ocurrirá un error de sintaxis si los dos símbolos en cualquiera de los operadores ==, !=, >= y <= están separados por espacios.
**2.17** Ocurrirá un error de sintaxis si los dos símbolos en cualquiera de los operadores, !=, >= y <= se invierten como en =!, =>, y =<, respectivamente.
**2.18** Confundir el operador de igualdad == con el operador de asignación =.
**2.19** Colocar un punto y coma de inmediato a la derecha del paréntesis derecho después de la condición en una estructura **if**.
**2.20** Colocar comas (cuando no se necesita ninguna) entre los especificadores de conversión en la cadena de control de formato en un enunciado **scanf**.

## *Prácticas sanas de programación*

**2.1** Todas las funciones deberán ser precedidas por un comentario que describa el objeto de la función.
**2.2** El último carácter impreso por una función que hace cualquier impresión, debería ser una nueva línea (\n). Esto asegura que la función dejará el cursor de pantalla colocado al principio de una nueva línea. Acuerdos de esta naturaleza estimulan la reutilización del software —una meta clave en los entornos de desarrollo de software.
**2.3** Haga una sangría de todo el cuerpo de cada función en un nivel (tres espacios) dentro de las llaves que definen el cuerpo de la función. Esto enfatiza la estructura funcional de los programas y ayuda a hacer que los programas sean más legibles.

2.4 Defina una regla convencional para el tamaño de la sangría que prefiera y a continuación aplíquela de manera uniforme. La tecla del tabulador puede ser utilizada para crear sangrías, pero los tabuladores pueden variar. Recomendamos utilizar ya sea tabuladores de 1/4 de pulgada o contar a mano tres espacios por cada uno de los niveles de sangría.

2.5 Aunque la inclusión de `<stdio.h>` es opcional, deberá ser incluida en cualquier programa C que utilice funciones de entrada/salida estándar de biblioteca. Esto ayuda al compilador a auxiliarle a usted a localizar errores en la fase de compilación de su programa, más bien que en la fase de ejecución (donde los errores resultan más costosos de corregir).

2.6. Coloque un espacio después de cada coma (,) para hacer los programas más legibles.

2.7 La selección de nombres de variables significativos ayuda a la autodocumentación de un programa, es decir, se requerirán de menos comentarios.

2.8 La primera letra de un identificador utilizado como un nombre simple de variable, deberá ser una letra minúscula. Más adelante en el texto le daremos significación especial a aquellos identificadores que empiezan con una letra mayúsculas y aquellos que utilizan todas mayúsculas.

2.9 Los nombres de variables de varias palabras pueden auxiliar a la legibilidad de un programa. Evite reunir las palabras separadas juntas como en `totalcommissions`. En vez de ello separe las palabras con subrayados como `total_commission`, o bien, si desea reunirlas, empiece cada palabra después del primero con una letra mayúscula como en `totalCommissions`.

2.10 Separe las declaraciones y los enunciados ejecutables en una función mediante una línea en blanco para enfatizar dónde terminan las declaraciones y dónde empiezan los enunciados ejecutables.

2.11 Coloque espacios en blanco a ambos lados de un operador binario. Esto hace resaltar al operador y permite que el programa sea mas legible.

2.12 Haga una sangría en el o los enunciados del cuerpo de una estructura `if`.

2.13 Coloque una línea en blanco antes y después de cada estructura de control en un programa para mejor legibilidad.

2.14 Deberá procurar que no exista más de un enunciado en cada línea de un programa.

2.15 Un enunciado largo puede ser dividido en varios renglones. Si un enunciado debe ser dividido en varias líneas, escoja puntos de corte que tengan sentido (como sería después de una coma en una lista separada por comas). Si un enunciado se divide en dos o más renglones, haga una sangría en todas las líneas subsecuentes.

2.16 Refiérase a la gráfica de precedencia de operadores al escribir expresiones que contengan muchos operadores. Confirme que los operadores en la expresión se ejecutan en el orden correcto. Si no está totalmente seguro del orden de evaluación en una expresión compleja, utilice paréntesis para obligar al orden, exactamente como lo haría en expresiones algebraicas. Asegúrese de observar que algunos de los operadores de C como el operador de asignación (=) se asocian de derecha a izquierda en vez de izquierda a derecha.

## *Sugerencia de portabilidad*

2.1 Utilice identificadores de 31 o menos caracteres. esto auxilia a asegurar la portabilidad y puede evitar algunos errores sutiles de programación.

## *Ejercicios de autoevaluación*

2.1 Llene cada uno de los siguientes espacios vacíos:

a) Cada programa en C empieza su ejecución en la función _____.

b) La _____ empieza el cuerpo de cada función y la _____ termina el cuerpo de cada función.

c) Cada enunciado termina con un _____

d) La función estándar de biblioteca _____ despliega información en la pantalla.



e) La secuencia de escape `\n` representa el carácter _____ que hace que se coloque el cursor en el principio de la siguiente línea en la pantalla.

f) La función estándar de biblioteca _____ se utiliza para obtener datos del teclado.

g) El especificador de conversión _____ se utiliza en una cadena de control de formato `scanf` para indicar que entrará un entero y en una cadena de control de formato `printf` para indicar que se sacará un entero.

h) Siempre que se coloca en una posición de memoria un nuevo valor, este valor borra el valor anterior en dicha posición. Este proceso se conoce como lectura _____.

i) Cuando se lee un valor desde una posición de memoria, el valor en dicha posición se conserva; esto se conoce como lectura _____.

j) El enunciado _____ se utiliza para tomar decisiones.

2.2 Indique si cada uno de los siguientes es verdadero o falso. Si es falso explique por qué.

a) Cuando la función `printf` es llamada ésta siempre empieza a imprimir en el principio de una nueva línea.

b) Los comentarios hacen que la computadora imprima el texto encerrado entre `/*` y `*/` en la pantalla al ejecutarse el programa.

c) La secuencia de escape `\n` cuando se utiliza en una cadena de control de formato `printf` hace que el cursor se coloque en el principio de la siguiente línea en pantalla.

d) Todas las variables deben ser declaradas antes de que puedan ser utilizadas.

e) Deben dársele un tipo a todas las variables cuando son declaradas.

f) C considera a las variables `number` y `NuMbEr` como idénticas.

g) Las declaraciones pueden aparecer en cualquier parte del cuerpo de una función.

h) Todos los argumentos que sigan a la cadena de control de formato en una función `printf` deben estar precedidas de un ampersand (`&`).

i) El operador de módulo (%) puede ser usado sólo con operandos enteros

j) Los operadores aritméticos `*`, `/`, `%`, `+`, y `-` tienen todos el mismo nivel de precedencia.

k) Verdadero o falso: los siguientes nombres de variable son idénticos en todos los sistemas de ANSI C.

```
thisisasuperduperlongname1234567
thisisasuperduperlongname1234568
```

l) Cierto o falso: un programa C que imprime tres líneas de salida debe contener tres enunciados `printf`

2.3 Escriba sólo un enunciado C para conseguir cada uno de los siguientes:

a) Declarar las variables `c`, `thisVariable`, `q76354`, y `number` para que sean del tipo `int`.

b) Indique al usuario para que escriba un entero. Termine su mensaje indicador con dos puntos (`:`), seguido por un espacio y deje el cursor colocado después de este espacio.

c) Lea un entero del teclado y almacene el valor escrito en una variable entera `a`.

d) Si la variable `number` no es igual a `7`, imprima `"The variable number is not equal to 7".`

e) Imprima el mensaje `"This is a C program."` sobre una sola línea.

f) Imprima el mensaje `"This is a C program."` sobre dos líneas donde la primera termine en C.

g) Imprima el mensaje `"This is a C program."` colocando cada palabra en una línea por separado.

h) Imprima el mensaje `"This is a C program."` con cada palabra separada por tabuladores.

2.4 Escriba un enunciado (o comentario) para cumplir con cada uno de lo siguiente:

a) Declare que un programa calculará el producto de tres enteros.

b) Declare las variables `x`, `y`, `z` y `result` que sean del tipo `int`.

c) Indíquele al usuario que escriba tres enteros.

d) Lea tres enteros del teclado y almacénelos en las variables `x`, `y`, `z`.

e) Calcule el producto de los tres enteros contenidos en las variables **x**, **y**, **z**, y asígnele el resultado a la variable `result`.
f) Imprima `"The product is"` seguido por el valor de la variable `result`.

**2.5** Utilizando los enunciados que escribió en el ejercicio 2.4, escriba un programa completo que calcule el producto de tres enteros.

**2.6** Identifique y corrija los errores de cada uno de los enunciados siguientes:

a)
```
printf ("The value is %d\n", &number);
```
b)
```
scanf("%d%d", &number1, number2);
```
c)
```
if (c < 7);
   printf("C is less than 7\n");
```
d)
```
if (c => 7)
   printf("C is equal to or less than 7\n");
```

## *Respuestas a los ejercicios de autoevaluación*

**2.1** a) `main`. b) llave izquierda ({), llave derecha (}). c)punto y coma. d) `printf`. e) nueva línea. f) `scanf`. g) %d. h) destructivo. i) no destructivo. j)`if`.

**2.2** a) Falso. La función `printf` siempre comienza la impresión en el lugar donde esté colocado el cursor y esto puede ser en cualquier parte de una línea sobre la pantalla.
b) Falso. Los comentarios no hacen que se realice acción alguna al ejecutarse el programa. Se utilizan para documentar programas y mejorar su legibilidad.
c) Verdadero.
d) Verdadero.
e) Verdadero.
f) Falso. C diferencia minúsculas de mayúsculas, por lo tanto, estas variables cada una es diferente y única.
g) Falso. Las declaraciones deben aparecer después de la llave izquierda del cuerpo de una función y antes de cualquier enunciado ejecutable.
h) Falso. Los argumentos en una función `printf` por lo regular no deberán ser precedidas por un ampersand. Los argumentos que siguen a la cadena de control de formato en una función `scanf` normalmente deberán ser antecedidos por un ampersand. Analizaremos las excepciones en el capítulo 6 y 7.
i) Verdadero.
j) Falso. Los operadores `*`, `/` y `%` están en un mismo nivel de precedencia, y los operadores + y -, están en un nivel inferior de precedencia,
k) Falso. Algunos sistemas pudieran distinguir entre identificadores más largos de 31 caracteres.
l) Falso. Un enunciado `printf` con varias secuencias de escape **n** puede imprimir varias líneas.

**2.3** a)
```
int c, thisVariable, q76354, number;
```
b)
```
printf("Enter an integer: ");
```
c)
```
scanf("%d", &a);
```
d)
```
if (number != 7)
   printf("The variable number is not equal to 7.\n");
```
e)
```
printf("This is a C program.\n");
```
f)
```
printf("This is a C\nprogram.\n");
```
g)
```
printf("This\nis\na\nC\nprogram.\n");
```
h)
```
printf("This\tis\ta\tC\tprogram.\n");
```

**2.4** a)
```
/* Calculate the product of three integers */
```
b)
```
int x, y, z, result;
```
c)
```
printf("Enter three integers: ");
```



d)
```
scanf("%d%d%d", &x, &y, &z);
```
e)
```
result = x * y * z;
```
f)
```
printf("The product is %d\n", result);
```

**2.5**
```
/* Calculate the product of three integers */
#include <stdio.h>

main()
{
  int x, y, z, result;

  printf("Enter three integers: ");
  scanf("%d%d%d", &x, &y, &z);
  result = x * y * z;
  printf("The product is %d\n", result);
  return 0;
}
```

**2.6** a) Error: `&number`. Corrección: eliminar el `&`. Más adelante en el texto analizaremos excepciones a esto.
b) Error: `number2` no tiene un ampersand. Corrección: `number2` debería decir `&number2`. Más adelante en el texto analizaremos excepciones a lo anterior.
c) Error: punto y coma después del paréntesis derecho de la condición en una enunciado `if`. Corrección: elimine el punto y coma después del paréntesis derecho. Nota: el resultado de este error es que el enunciado `printf` se ejecutará independiente de que la condición en el enunciado `if` resulte verdadera. El punto y coma después del paréntesis derecho se considera como un enunciado vacío —un enunciado que no hace nada.
d) Error: El operador relacional => debe de ser cambiado por >=.

## *Ejercicios*

**2.7** Identifique y corrija los errores en cada uno de los siguientes enunciados (Nota: pudieran existir más de un error por cada enunciado):

a)
```
scanf("d", value);
```
b)
```
printf("The product of %d and %d is %d"\n, x, y);
```
c)
```
firstNumber + secondNumber = sumOfNumbers
```
d)
```
if (number => largest)
   largest == number;
```
e)
```
*/ Program to determine the largest of three integers /*
```
f)
```
Scanf("%d", anInteger);
```
g)
```
printf("Remainder of %d divided by %d is\n", x, y, x % y);
```
h)
```
if (x = y);
   printf(%d is equal to %d\n", x, y);
```
i)
```
printf("The sum is %d\n," x + y);
```
j)
```
Printf ("The value you entered is: %d\n, &value);
```

**2.8** Llene los espacios vacíos en cada uno de los siguientes:
a) _____ se utilizan para comentar un programa y mejorar su legibilidad.
b) La función utilizada para imprimir información en la pantalla es _____.
c) Un enunciado C que toma una decisión es _____.
d) Los cálculos por lo regular se ejecutan por enunciados _____.
e) La función _____ introduce valores del teclado.

**2.9** Escriba solo un enunciado de C o una línea que cumpla con cada uno de lo siguiente:
a) Imprima el mensaje `"Enter two numbers."`
b) Asigne el producto de las variables **b** y **c** a la variable **a**.
c) Declare que un programa ejecuta un cálculo de muestra de nómina (es decir, utiliza texto que auxilia a documentar un programa).
d) Introduzca dos valores enteros del teclado y coloque estos valores en las variables enteras **a**, **b** y **c**.

**2.10** Declare cuales de los siguientes son verdaderos y cuales son falsos. Explique sus respuestas.
a) Los operadores C se evalúan de izquierda a derecha.
b) Los siguientes son todos nombres válidos de variable: `_under_bar_`, `m928134`, `t5`, `j7`, `her_sales`, `his_account_total`, `a`, `b`, `c`, `z`, `z2`.
c) El enunciado `printf("a = 5;");` es un ejemplo típico de un enunciado de asignación.
d) Una expresión aritmética válida en C que no contenga paréntesis se evalúa de izquierda a derecha.
e) Lo siguientes son todos nombres inválidos de variables: `3g`, `87`, `67h2`, `h22`, `2h`.

**2.11** Llene los espacios vacíos en cada uno de los siguientes:
a) ¿Qué operaciones aritméticas tienen el mismo nivel de precedencia que la multiplicación? _____.
b) ¿Cuando los paréntesis están anidados, qué conjunto de paréntesis serán calculados en primer término en una expresión aritmética? _____.
c) Una posición en la memoria de la computadora que puede contener valores diferentes en tiempos diferentes a lo largo de la ejecución de un programa se conoce como _____.

**2.12.** ¿Qué es lo que, si es que algo, se imprime cuando se ejecutan cada uno de los enunciados de C siguientes? Si no se imprime nada, entonces conteste "nada". Suponga que `x = 2 y y =3`

a) `printf("%d", x);`
b) `printf("%d", x + x);`
c) `printf("x=");`
d) `printf("x=%d", x);`
e) `printf("%d = %d", x + y, y + x);`
f) `z = x + y;`
g) `scanf ("%d%d", &x, &y);`
h) `/* printf("x + y = %d", x + y); */`
i) `printf ("\n");`

**2.13** ¿Qué es, si es que es algo, de los enunciados siguientes contienen variables involucradas en lecturas destructivas?.

a) `scanf("%d%d%d%d%d", &b, &c, &d, &e, &f);`
b) `p = i + j +k + 7;`
c) `printf("Destructive read-in");`
d) `printf("a = 5");`

**2.14** Dada la ecuación $y = ax^3 + 7$, ¿cuál de los que siguen, si es que existe alguno, son enunciados correctos de C correspondientes a esta ecuación?.

a) `y = a * x * x * x + 7;`
b) `y = a * x * x * (x + 7);`
c) `y = (a * x) * x * (x + 7);`
d) `y = (a * x) * x * x + 7;`
e) `y = a * (x * x * x) + 7;`
f) `y = a * x * (x * x + 7);`

**2.15** Declare el orden de cálculo de los operadores en cada uno de los enunciados de C siguientes, y muestre el valor de **x** después de que se ejecute cada uno de ellos.



a) `x = 7 + 3 * 6 / 2 -1;`
b) `x = 2 % 2 + 2 * 2 - 2 / 2;`
c) `x = (3 * 9 * (3 + (9 * 3/ (3))));`

**2.16** Escriba un programa que solicite al usuario que introduzca dos números, tome los dos números del usuario, e imprima la suma, el producto, la diferencia, el cociente y el módulo de los dos números.

**2.17** Escriba un programa que imprima los números 1 a 4 en un mismo renglón. Escriba el programa utilizando los siguientes métodos.
a) Utilizando un enunciado `printf` sin especificadores de conversión.
b) Utilizando un enunciado `printf` con cuatro especificadores de conversión.
c) Utilizando cuatro enunciados `printf`.

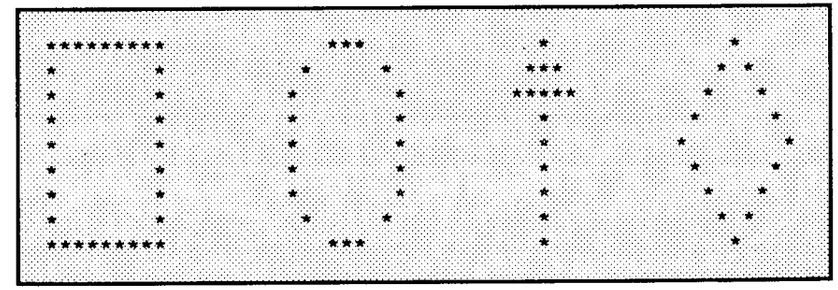
**2.18** Escriba un programa que solicite al usuario que escriba dos enteros, tome los números del usuario y a continuación imprima el número mayor seguido por las palabras `"is larger"`. Si los números son iguales, que imprima el mensaje `"These numbers are equal"`. Utilice sólo la forma de una selección del enunciado `if` que aprendió en este capítulo.

**2.19** Escriba un programa de C que entre tres enteros diferentes del teclado, y a continuación imprima la suma, el promedio, el producto, el más pequeño y el más grande de estos números. Utilice sólo la forma de una selección del enunciado `if`, que usted aprendió en este capítulo. El diálogo en pantalla deberá aparecer como sigue:

```
Input three different integers: 13 27 14
Sum is 54
Average is 18
Product is 4914
Smallest is 13
Largest is 27
```

**2.20** Escriba un programa que lea el radio de un círculo y que imprima el diámetro del mismo, su circunferencia y su área.Utilice el valor constante 3.14159 para "pi". Efectúe cada uno de estos cálculos dentro del enunciado o enunciados `printf` y utilice el especificador de conversión `%f`. (Nota: en este capítulo, hemos estudiado únicamente constantes y variables enteras. En el capítulo 3 veremos números de punto flotante, es decir, valores que pueden tener puntos decimales).

**2.21** Escriba un programa que imprima un recuadro, un oval, una flecha y un diamante, como sigue:



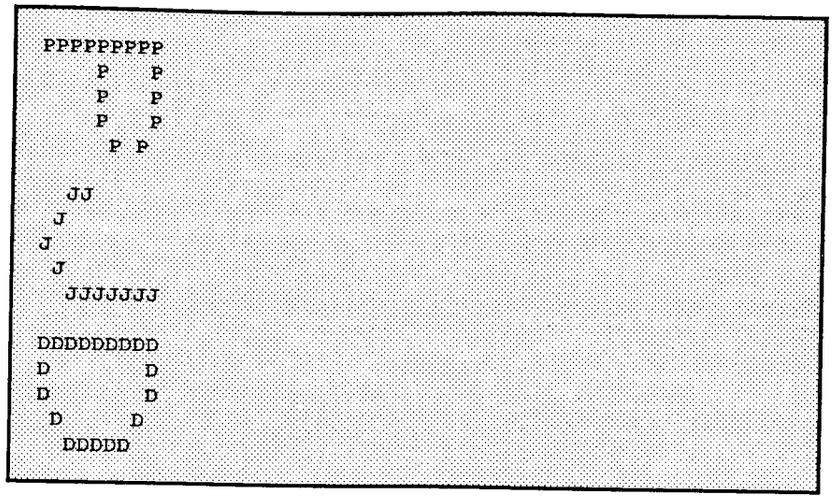
**2.22** ¿Qué es lo que imprime el código siguiente?

```
printf("*\n**\n***\n****\n*****\n");
```

**2.23** Escriba un programa que lea cinco enteros y a continuación determine e imprima cuáles son el mayor y el menor entero en el grupo. Utilice sólo las técnicas de programación que aprendió en este capítulo.

**2.24** Escriba un programa que lea un entero y determine e imprima si es par o impar. (*Sugerencia:* utilice el operador de módulo. Un número par es un múltiplo de dos. Cualquier múltiplo de dos deja un residuo de cero al ser dividido entre dos).

**2.25** Imprima sus iniciales en letras de bloque hacia abajo de la página. Construya cada letra de bloque utilizando las letras que representa como sigue:



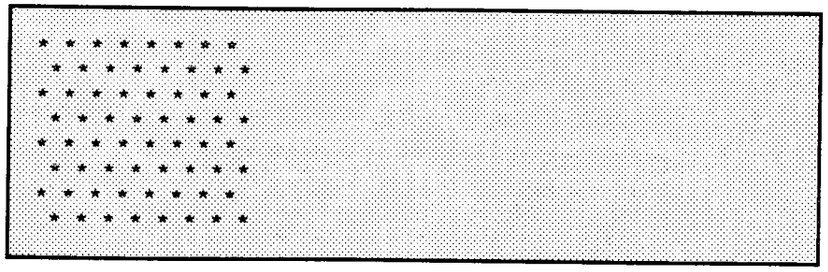
```
PPPPPPPPP
    P    P
    P    P
    P    P
     P P

  JJ
 J
J
 J
  JJJJJJJ

DDDDDDDDD
D        D
D        D
 D      D
  DDDDD
```

**2.26** Escriba un programa que lea dos enteros y que determine e imprima si el primero es un múltiplo del segundo. (*Sugerencia:* utilice el operador de módulo).

**2.27** Despliegue un patrón cuadriculado utilizando ocho enunciados `printf`, y a continuación despliegue el mismo patrón con el mínimo posible de enunciados `printf`.



```
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
* * * * * * * *
 * * * * * * * *
```

**2.28** Distinga entre los términos error fatal y error no fatal. ¿Por qué preferiría experimentar un error fatal en vez de un error no fatal?

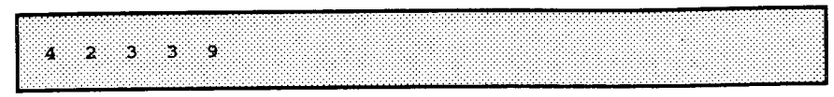
**2.29** He aquí una mirada hacia adelante. En este capítulo usted aprendió los enteros y el tipo `int`. C también puede representar letras mayúsculas, letras minúsculas y una gran variedad de símbolos especiales. C utiliza enteros pequeños en forma interna para representar cada carácter diferente. El conjunto de caracteres que utiliza una computadora y las representaciones de entero correspondientes para estos caracteres se conoce como el conjunto de caracteres de la computadora. Por ejemplo, usted puede imprimir el equivalente en enteros de la `A`, ejecutando el enunciado

```
printf("%d", 'A');
```



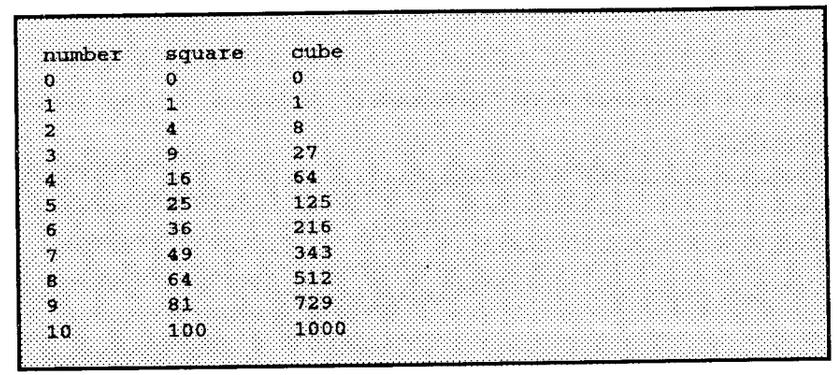
Escriba un programa C que imprima los equivalentes en enteros de algunas letras mayúsculas, letras minúsculas, dígitos y símbolos especiales. Como mínimo, determine los equivalentes en enteros de los siguientes: `A B C a b c 0 1 2 $ * + /` así como del carácter de espacio en blanco.

**2.30** Escriba un programa que entre un número de cinco dígitos, separe el número en sus dígitos individuales e imprima los dígitos separados unos de otros mediante tres espacios. Por ejemplo, si el usuario escribe `42339` el programa debería imprimir



```
4   2   3   3   9
```

**2.31** Utilizando sólo las técnicas aprendidas en este capítulo, escriba un programa que calcule los cuadrados y los cubos de los números del 1 al 10 y que utilice tabuladores para imprimir la siguiente tabla de valores:



| number | square | cube |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 4 | 8 |
| 3 | 9 | 27 |
| 4 | 16 | 64 |
| 5 | 25 | 125 |
| 6 | 36 | 216 |
| 7 | 49 | 343 |
| 8 | 64 | 512 |
| 9 | 81 | 729 |
| 10 | 100 | 1000 |