

Una revisión de las computadoras y la programación

En este capítulo usted aprenderá sobre:

- ⦿ Sistemas de cómputo
- ⦿ Lógica de un programa simple
- ⦿ Los pasos que se siguen en el ciclo de desarrollo del programa
- ⦿ Declaraciones de pseudocódigo y símbolos de diagramas de flujo
- ⦿ Usar un valor centinela para terminar un programa
- ⦿ Programación y ambientes de usuario
- ⦿ La evolución de los modelos de programación

Comprensión de los sistemas de cómputo

Un **sistema de cómputo** es una combinación de todos los componentes que se requieren para procesar y almacenar datos usando una computadora. Todos los sistemas de cómputo están compuestos por múltiples piezas de hardware y software.

- **Hardware** es el equipo o los dispositivos físicos asociados con una computadora. Por ejemplo, todos los teclados, ratones, altavoces e impresoras son hardware. Los dispositivos se manufacturan en forma diferente para las computadoras mainframe grandes, las laptops e incluso para las computadoras más pequeñas que están incorporadas en los productos como automóviles y termostatos, pero los tipos de operaciones que efectúan las computadoras de distintos tamaños son muy parecidos. Cuando se piensa en una computadora con frecuencia son sus componentes físicos los que llegan a la mente, pero para que sea útil necesita más que dispositivos; requiere que se le den instrucciones. Del mismo modo en que un equipo de sonido no hace mucho hasta que se le incorpora la música, el hardware de computadora necesita instrucciones que controlen cómo y cuándo se introducen los datos, cómo se procesan y la forma en que se les da salida o se almacenan.
- **Software** son las instrucciones de la computadora que dicen al hardware qué hacer. El software son **programas** o conjuntos de instrucciones escritos por programadores. Usted puede comprar programas previamente escritos que se han almacenado en un disco o descargarlos de la web. Por ejemplo, en los negocios se utilizan programas de procesamiento de palabras y de contabilidad y los usuarios ocasionales de computadoras disfrutan los que reproducen música y juegos. De manera alternativa, usted puede escribir sus propios programas. Cuando escribe las instrucciones de un software se dice que está **programando**. Este libro se enfoca en el proceso de programación.

El software puede clasificarse en dos extensas categorías:

- **Software de aplicación**, que abarca todos los programas que se aplican para una tarea, como los procesadores de palabras, las hojas de cálculo, los programas de nómina e inventarios, e incluso los juegos.
- **Software de sistema**, que incluye los programas que se usan para manejar una computadora, entre los que se encuentran los sistemas operativos como Windows, Linux o UNIX.

Este libro se enfoca en la lógica que se usa para escribir programas de software de aplicación, aunque muchos de los conceptos se aplican a ambos tipos de software.

Juntos, el hardware y el software llevan a cabo tres operaciones importantes en la mayoría de los programas:

- **Entrada:** los elementos de datos entran en el sistema de cómputo y se colocan en la memoria, donde pueden ser procesados. Los dispositivos de hardware que efectúan operaciones de entrada incluyen teclados y ratones. Los **elementos de datos** constan de todo el texto, los números y otras materias primas que se introducen en una computadora y son procesadas por ella. En los negocios, muchos elementos de datos que se usan son los hechos y las cifras sobre entidades como productos, clientes y personal. Sin embargo, los datos también pueden incluir imágenes, sonidos y movimientos del ratón que el usuario efectúa.
- **Procesamiento:** procesar los elementos de datos implica organizarlos o clasificarlos, comprobar su precisión o realizar cálculos con ellos. El componente de hardware que realiza estas tareas es la **unidad central de procesamiento** o **CPU** (siglas del inglés *central processing unit*).

- **Salida:** después de que los elementos de datos se han procesado, la información resultante por lo general se envía a una impresora, un monitor o algún otro dispositivo de salida de modo que las personas vean, interpreten y usen los resultados. Los profesionales de la programación con frecuencia emplean el término *datos* para los elementos de entrada y el de **información** para los datos que se han procesado y que han salido. En ocasiones usted coloca estos datos de salida en **dispositivos de almacenamiento**, como discos o medios flash (*flash media*). Las personas no pueden leer los datos en forma directa desde tales dispositivos, pero éstos contienen información que puede recuperarse posteriormente. Cuando se envía una salida a un dispositivo de almacenamiento en ocasiones se usa después como entrada para otro programa.

Las instrucciones para el ordenador se escriben en un **lenguaje de programación** como Visual Basic, C#, C++ o Java. Del mismo modo en que algunas personas hablan inglés y otras japonés, los programadores escriben en diferentes lenguajes. Algunos de ellos trabajan de manera exclusiva en uno de ellos mientras que otros conocen varios y usan aquel que sea más adecuado para la tarea que se les presenta.

Las instrucciones que usted escribe usando un lenguaje de programación constituyen un **código de programa**; cuando las escribe está **codificando el programa**.

Cada lenguaje de programación tiene reglas que rigen el uso de las palabras y la puntuación. Estas reglas se llaman **sintaxis** del lenguaje. Los errores en el uso de un lenguaje son **errores de sintaxis**. Si usted pregunta: “¿Cómo otengo forma la guardarlo de?” en español, casi todas las personas pueden imaginar lo que probablemente usted quiso decir, aun cuando no haya usado una sintaxis apropiada en español: ha mezclado el orden de las palabras y ha escrito mal una de ellas. Sin embargo, las computadoras no son tan inteligentes como la mayoría de las personas; en este caso, bien podría haber preguntado a la computadora: “¿Xpu mxv ort dod nmcad bf B?”. A menos que la sintaxis sea perfecta, la computadora no puede interpretar en absoluto la instrucción del lenguaje de programación.

Cuando usted escribe un programa por lo general transmite sus instrucciones usando un teclado. Cuando lo hace, las instrucciones se almacenan en la **memoria de la computadora**, que es un almacenamiento interno temporal. La **memoria de acceso aleatorio** o **RAM** es una forma de memoria interna volátil. Los programas que “corren” (es decir, se ejecutan) en la actualidad y los elementos de datos que se usan se almacenan en la RAM para que sea posible tener un acceso rápido a ellos. El almacenamiento interno es **volátil**, es decir, su contenido se pierde cuando la computadora se apaga o se interrumpe la energía. Por lo general, usted desea recuperar y quizá modificar más tarde las instrucciones almacenadas, de modo que también tiene que guardarlas en un dispositivo de almacenamiento permanente, como un disco. Estos dispositivos se consideran **no volátiles**, esto es, su contenido es permanente y se conserva aun cuando la energía se interrumpa. Si usted ha experimentado una interrupción de la energía mientras trabajaba en una computadora pero pudo recuperar su información cuando aquélla se restableció, no se debió a que su trabajo todavía se encontrara en la RAM. Su sistema se ha configurado para guardar automáticamente su trabajo a intervalos regulares en un dispositivo de almacenamiento no volátil.

Después de que se ha escrito un programa usando declaraciones de un lenguaje de programación y se ha almacenado en la memoria, debe traducirse a un **lenguaje de máquina** que representa los millones de circuitos encendidos/apagados dentro de la computadora. Sus declaraciones en lenguaje de programación se llaman **código fuente** y las traducidas al lenguaje de máquina se denominan **código objeto**.

Cada lenguaje de programación usa una pieza de software llamada **compilador** o **intérprete** para traducir su código fuente al lenguaje de máquina. Este último también se llama **lenguaje binario** y se representa como una serie de 0 (ceros) y 1 (unos). El compilador o intérprete que traduce el código indica si cualquier componente del lenguaje de programación se ha usado de manera incorrecta. Los errores de sintaxis son relativamente fáciles de localizar y corregir debido a que el compilador o intérprete los resalta. Si usted escribe un programa de computadora usando un lenguaje como C++ pero deletrea en forma incorrecta una de sus palabras o invierte el orden apropiado de dos de ellas, el software le hace saber que encontró un error desplegando un mensaje tan pronto como usted intenta traducir el programa.



Aunque hay diferencias en la forma en que trabajan los compiladores y los intérpretes, su función básica es la misma: traducir sus declaraciones de programación en un código que la computadora pueda usar. Cuando usted usa un compilador, un programa entero se traduce antes de que pueda ejecutarse; cuando utiliza un intérprete, cada instrucción es traducida justo antes de la ejecución. Por lo general usted no elige cuál tipo de traducción usar, esto depende del lenguaje de programación. Sin embargo, hay algunos lenguajes que disponen tanto de compiladores como de intérpretes.

Después de que el código fuente es traducido con éxito al lenguaje de máquina, la computadora puede llevar a cabo las instrucciones del programa. Un programa **corre** o se **ejecuta** cuando se realizan las instrucciones. En un programa típico se aceptará alguna entrada, ocurrirá algún procesamiento y se producirá alguna salida.



Además de los lenguajes de programación exhaustivos populares como Java y C++, muchos programadores usan **lenguajes de programación interpretados** (que también se llaman **lenguajes de programación de scripting** o **lenguajes de script**) como Python, Lua, Perl y PHP. Los scripts escritos en estos lenguajes por lo general pueden mecanografiarse en forma directa desde un teclado y almacenarse como texto en lugar de como archivos ejecutables binarios. Los lenguajes de script son interpretados línea por línea cada vez que se ejecuta el programa, en lugar de ser almacenados en forma compilada (binaria). Aun

DOS VERDADES Y UNA MENTIRA

Comprensión de los sistemas de cómputo

En cada sección “Dos verdades y una mentira”, dos de las afirmaciones numeradas son verdaderas y una es falsa. Identifique la que es falsa y explique por qué lo es.

1. El hardware es el equipo o los dispositivos asociados con una computadora. El software son las instrucciones.
2. Las reglas gramaticales de un lenguaje de programación de computadoras constituyen su sintaxis.
3. Usted escribe programas usando el lenguaje de máquina y el software de traducción convierte las declaraciones a un lenguaje de programación.

La afirmación falsa es la número 3. Se escriben programas usando un lenguaje de programación como Visual Basic o Java, y un programa de traducción (llamado compilador o intérprete) convierte las declaraciones en lenguaje de máquina, el cual se compone de 0 (ceros) y 1 (unos).

así, con todos los lenguajes de programación cada instrucción debe traducirse al lenguaje de máquina antes de que pueda ejecutarse.

Comprensión de la lógica de programa simple

Un programa con errores de sintaxis no puede traducirse por completo ni ejecutarse. Uno que no los tenga es traducible y puede ejecutarse, pero todavía podría contener **errores lógicos** y dar como resultado una salida incorrecta. Para que un programa funcione en forma apropiada usted debe desarrollar una **lógica** correcta, es decir, escribir las instrucciones en una secuencia específica, no dejar fuera ninguna instrucción y no agregar instrucciones ajenas.

Suponga que indica a alguien que haga un pastel de la siguiente manera:

Consiga un tazón
Revuelva
Agregue dos huevos
Agregue un litro de gasolina
Hornee a 350° por 45 minutos
Agregue tres tazas de harina

No lo haga
¡No hornee un pastel
como éste!



Las instrucciones peligrosas para hornear un pastel se muestran con un icono “No lo haga”. Verá este icono cuando se presente en el libro una práctica de programación no recomendada que se usa como ejemplo de lo que *no* debe hacerse.

Aun cuando la sintaxis de las instrucciones para hornear un pastel es correcta en español, están fuera de secuencia; faltan algunas y otras pertenecen a procedimientos distintos que no tienen nada que ver con hornear un pastel. Si las sigue no hará un pastel que sea posible ingerir y quizá termine por ser un desastre. Muchos errores lógicos son más difíciles de localizar que los de sintaxis; es más fácil determinar si la palabra “huevos” en una receta está escrita en forma incorrecta que decir si hay demasiados o si se agregaron demasiado pronto.

Del mismo modo en que las instrucciones para hornear pueden proporcionarse en chino mandarín, urdu o inglés, la lógica de programa puede expresarse en forma correcta en cualquier cantidad de lenguajes de programación. Debido a que este libro no se enfoca en algún lenguaje específico, los ejemplos de programación pudieron escribirse en Visual Basic, C++ o Java. Por conveniencia, ¡en este libro las instrucciones se han escrito en español!



Después de aprender francés, usted automáticamente conoce, o puede imaginar con facilidad, muchas palabras en español. Del mismo modo, después de aprender un lenguaje de programación, es mucho más fácil entender otros lenguajes.

Los programas de computadora más sencillos incluyen pasos que ejecutan la entrada, el procesamiento y la salida. Suponga que desea escribir un programa para duplicar cualquier número que proporcione. Puede escribirlo en un lenguaje como Visual Basic o Java, pero si lo escribiera con declaraciones en inglés, se verían así:

```
input myNumber
set myAnswer = myNumber * 2
output myAnswer
```

El proceso de duplicar el número incluye tres instrucciones:

- La instrucción `input myNumber` es un ejemplo de una operación de entrada. Cuando la computadora interpreta esta instrucción sabe que debe buscar un dispositivo de entrada para obtener un número. Cuando usted trabaja en un lenguaje de programación específico, escribe instrucciones que indican a la computadora a cuál dispositivo se tiene acceso para obtener la entrada. Por ejemplo, cuando un usuario introduce un número como los datos para un programa podría hacer clic en el número con un ratón, mecanografiarlo en un teclado o hablar en un micrófono. Sin embargo, es lógico que no importa cuál dispositivo de hardware se use siempre que la computadora sepa que debe aceptar un número. Cuando el número se recupera desde un dispositivo de entrada, se coloca en la memoria de la computadora en una variable llamada `myNumber`. Una **variable** es una ubicación de memoria nombrada cuyo valor puede variar; por ejemplo, el valor de `myNumber` podría ser 3 cuando el programa se usa por primera vez y 45 cuando se usa la siguiente vez. En este libro, los nombres de las variables no llevarán espacios; por ejemplo, se usará `myNumber` en lugar de `my Number`.



Desde una perspectiva lógica, cuando usted introduce, procesa o da salida a un valor, el dispositivo de hardware es irrelevante. Lo mismo sucede en su vida diaria. Si sigue la instrucción “Obtener huevos para el pastel”, en realidad no importa si los compra en una tienda o los recolecta de sus propias gallinas; usted los consigue de cualquier forma. Podría haber diferentes consideraciones prácticas para obtenerlos, del mismo modo que las hay para obtener los datos de una base de datos grande en contraposición a obtenerlos de un usuario inexperto que trabaja en casa en una laptop. Por ahora, este libro sólo se interesa en la lógica de las operaciones, no en detalles menores.

- La instrucción `set myAnswer = myNumber * 2` es un ejemplo de una operación de procesamiento. En la mayoría de los lenguajes de programación se usa un asterisco para indicar una multiplicación, de modo que esta instrucción significa “Cambiar el valor de la ubicación de memoria `myAnswer` para igualar el valor en la ubicación de memoria `myNumber` por dos”. Las operaciones matemáticas no son el único tipo de operaciones de procesamiento, pero son típicas. Como sucede con las operaciones de entrada, el tipo de hardware que se usa para el procesamiento es irrelevante; después de que usted escribe un programa, éste puede usarse en computadoras de diferentes marcas, tamaños y velocidades.
- En el programa para duplicar un número, la instrucción `output myAnswer` es un ejemplo de una operación de salida. Dentro de un programa particular, esta declaración podría causar que la salida aparezca en el monitor (digamos, una pantalla plana de plasma o una de tubo de rayos catódicos), que vaya a una impresora (láser o de inyección de tinta) o que se escriba en un disco o un DVD. La lógica del proceso de salida es la misma sin importar qué dispositivo de hardware se use. Cuando se ejecuta esta instrucción, el valor almacenado en la memoria en la ubicación llamada `myAnswer` se envía a un dispositivo de salida. (El valor de salida también permanece en la memoria hasta que se almacena algo más en la misma ubicación o se interrumpe la energía eléctrica.)



La memoria de la computadora consiste en millones de ubicaciones numeradas donde es posible almacenar datos. La ubicación de memoria de **myNumber** tiene una ubicación numérica específica, pero cuando usted escribe programas rara vez necesita preocuparse por el valor de la dirección de memoria; en cambio, usa el nombre fácil de recordar que creó. Los programadores de computadoras con frecuencia se refieren a las direcciones de memoria usando la notación hexadecimal, o en base 16. Con este sistema podrían utilizar un valor como 42FF01A para referirse a una dirección de memoria. A pesar del uso de letras, dicha dirección todavía es un número hexadecimal. El apéndice A contiene información sobre este sistema de numeración.

DOS VERDADES Y UNA MENTIRA

Comprensión de la lógica de programa simple

1. Un programa con errores de sintaxis puede ejecutarse pero podría generar resultados incorrectos.
2. Aunque la sintaxis de los lenguajes de programación difiere, la misma lógica de programa puede expresarse en diferentes lenguajes.
3. Los programas de computadora más sencillos incluyen pasos que efectúan entrada, procesamiento y salida.

La afirmación falsa es la número 1. Un programa con errores de sintaxis no puede ejecutarse; uno que no tenga este tipo de errores puede ejecutarse, pero produciría resultados incorrectos.

Comprensión del ciclo de desarrollo del programa

El trabajo de un programador implica escribir instrucciones (como las del programa para duplicar números en la sección anterior), pero por lo general un profesional no sólo se sienta ante un teclado de computadora y comienza a mecanografiar. La figura 1-1 ilustra el **ciclo de desarrollo del programa**, que se divide al menos en siete pasos:

1. Entender el problema.
2. Planear la lógica.
3. Codificar el programa.
4. Usar software (un compilador o intérprete) para traducir el programa a lenguaje de máquina.
5. Probar el programa.
6. Poner el programa en producción.
7. Mantener el programa.

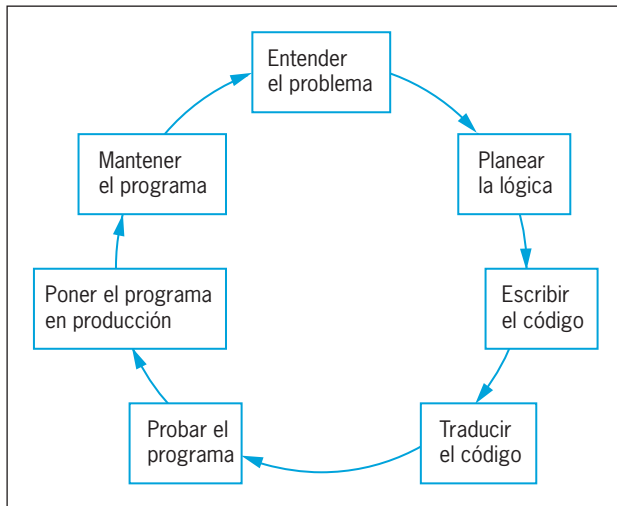


Figura 1-1 El ciclo de desarrollo del programa

Entender el problema

Los programadores profesionales escriben programas para satisfacer las necesidades de otras personas, llamadas **usuarios** o **usuarios finales**. Entre los ejemplos de usuarios finales estaría un departamento de recursos humanos que necesita una lista impresa de todos los empleados, un área de facturación que desea un listado de los clientes que se han retrasado en sus pagos 30 días o más, o un departamento de pedidos que requiere un sitio web para proporcionar a los compradores un carrito de compras en línea. Debido a que los programadores brindan un servicio a estos usuarios deben comprender primero lo que éstos desean. Cuando usted corre un programa con frecuencia piensa en la lógica como un ciclo de operaciones de entrada-procesamiento-salida; pero cuando planea un programa piensa primero en la salida. Después de entender cuál es el resultado deseado puede planear los pasos de entrada y procesamiento para lograrlo.

Suponga que el director de recursos humanos dice a un programador: “Nuestro departamento necesita una lista de todos los empleados que han estado aquí por más de cinco años, porque queremos invitarlos a una cena especial de agradecimiento”. En apariencia esta es una solicitud sencilla. Sin embargo, un programador experimentado sabrá que la solicitud está incompleta. Por ejemplo, quizá no sepa las respuestas a las siguientes preguntas sobre cuáles empleados incluir:

- ¿El director desea una lista sólo de empleados de tiempo completo o de tiempo completo y de medio tiempo juntos?
- ¿Desea incluir personas que han trabajado para la compañía con una base contractual mensual durante los pasados cinco años o sólo los empleados permanentes regulares?
- ¿Los empleados necesitan haber trabajado para la organización por cinco años hasta el día de hoy, hasta la fecha de la cena o en alguna otra fecha límite?
- ¿Qué pasa con un empleado que trabajó tres años, tomó una licencia de dos años y volvió a trabajar por tres años?

El programador no puede tomar ninguna de estas decisiones; el usuario (en este caso, el director de recursos humanos) debe abordar estas preguntas.

Tal vez aún se requiera tomar otras decisiones, por ejemplo:

- ¿Qué datos deben incluirse para cada empleado en la lista? ¿Es preciso anotar el nombre y los apellidos? ¿Los números de seguro social? ¿Números telefónicos? ¿Direcciones?
- ¿La lista debe estar en orden alfabético? ¿Por número de identificación del empleado? ¿En orden de años de servicio? ¿Algún otro orden?
- ¿Los empleados deberían agruparse con algún criterio, como número de departamento o años de servicio?

A menudo se proporcionan algunas piezas de documentación para ayudar al programador a entender el problema. La **documentación** consiste en todo el papeleo de soporte para un programa; podría incluir elementos como las solicitudes originales para el programa de los usuarios, muestras de salida y descripciones de los elementos de datos disponibles para la entrada.

Entender por completo el problema es uno de los aspectos más difíciles de la programación. En cualquier trabajo, la descripción de lo que el usuario necesita puede ser imprecisa; peor aún, los usuarios quizá no sepan qué desean en realidad, y los que piensan que saben a menudo cambian de opinión después de ver una muestra de salida. ¡Un buen programador es en parte consejero y en parte detective!

Planear la lógica

El corazón del proceso de programación se encuentra en la planeación de la lógica del programa. Durante esta fase, el programador planifica los pasos del mismo, decidiendo cuáles incluir y cómo ordenarlos. Usted puede visualizar la solución de un problema de muchas maneras. Las dos herramientas de programación más comunes son los diagramas de flujo y elseudocódigo; ambas implican escribir los pasos del programa en inglés, del mismo modo en que planearía un viaje en papel antes de subirse al automóvil o el tema de una fiesta antes de comprar alimentos y recuerdos.

Quizá usted haya escuchado a los programadores referirse a la planeación de un programa como “desarrollar un algoritmo”. Un **algoritmo** es la secuencia de pasos necesarios para resolver cualquier problema.



Además de los diagramas de flujo y elseudocódigo, los programadores usan una variedad de herramientas distintas para desarrollar el programa. Una de ellas es la **gráfica IPO**, que define las tareas de entrada, procesamiento y salida. Algunos programadores orientados hacia los objetos también usan **gráficas TOE**, que listan tareas, objetos y eventos.

El programador no debe preocuparse por la sintaxis de algún lenguaje en particular durante la etapa de planeación, sino enfocarse en averiguar qué secuencia de eventos llevará desde la entrada disponible hasta la salida deseada. La planeación de la lógica incluye pensar con

cuidado en todos los valores de datos posibles que un programa podría encontrar y cómo desea que éste maneje cada escenario. El proceso de recorrer en papel la lógica de un programa antes de escribirlo en realidad se llama **prueba de escritorio** (*desk-checking*). Aprenderá más sobre la planeación de la lógica a lo largo de este libro; de hecho, éste se enfoca casi de manera exclusiva en este paso crucial.

Codificación del programa

Sólo después de que se ha desarrollado la lógica el programador puede escribir el código fuente. Hay cientos de lenguajes de programación disponibles. Los programadores eligen lenguajes particulares debido a que algunos incorporan capacidades que los hacen más eficientes que otros para manejar ciertos tipos de operaciones. A pesar de sus diferencias, los lenguajes de programación son bastante parecidos en sus capacidades básicas; cada uno puede manejar operaciones de entrada, procesamiento aritmético, operaciones de salida y otras funciones estándares. La lógica que se desarrolla para resolver un problema de programación puede ejecutarse usando cualquier cantidad de lenguajes. Sólo después de elegir alguno el programador debe preocuparse por la puntuación y la ortografía correctas de los comandos; en otras palabras, por usar la *sintaxis* correcta.

Algunos programadores experimentados combinan con éxito en un paso la planeación de la lógica y la codificación del programa. Esto funciona para planear y escribir un programa muy sencillo, del mismo modo en que usted puede planear y escribir una postal para un amigo en un solo paso. Sin embargo, la redacción de un buen ensayo semestral o un guión cinematográfico requiere planeación y lo mismo sucede con la mayor parte de los programas.

¿Cuál paso es más difícil: planear la lógica o codificar el programa? Ahora mismo quizá le parezca que escribir en un lenguaje de programación es una tarea muy difícil, considerando todas las reglas de ortografía y sintaxis que debe aprender. Sin embargo, en realidad el paso de planeación es más difícil. ¿Qué es más complicado: pensar en los giros de la trama de una novela de misterio que es un éxito de ventas o escribir la traducción del inglés al español de una novela que ya se ha escrito? ¿Y quién cree que recibe más paga, el escritor que crea la trama o el traductor? (¡Haga la prueba pidiendo a algunos amigos que nombren a algún traductor famoso!)

Uso de software para traducir el programa al lenguaje de máquina

Aun cuando hay muchos lenguajes de programación, cada computadora conoce sólo uno: su lenguaje de máquina, que consiste en 1 (unos) y 0 (ceros). Las computadoras entienden el lenguaje de máquina porque están formadas por miles de diminutos interruptores eléctricos, cada uno de los cuales presenta un estado de encendido o apagado, que se representa con 1 o 0, respectivamente.

Lenguajes como Java o Visual Basic están disponibles para los programadores debido a que alguien ha escrito un programa traductor (un compilador o intérprete) que cambia el **lenguaje de programación de alto nivel** en inglés del programador en un **lenguaje de máquina de bajo nivel** que la computadora entiende. Cuando usted aprende la sintaxis de un lenguaje de programación, los comandos funcionan en cualquier máquina en la que el software del lenguaje se haya instalado. Sin embargo, sus comandos son traducidos entonces al lenguaje de máquina, que es distinto en las distintas marcas y modelos de computadoras.

Si usted escribe en forma incorrecta una declaración de programación (por ejemplo, escribe mal una palabra, usa alguna que no existe o utiliza gramática “ilegal”), el programa traductor no sabe cómo proceder y emite un mensaje al detectar un error de sintaxis. Aunque nunca es deseable cometerlos, los errores de sintaxis no son una preocupación importante para los programadores porque el compilador o intérprete los detecta y muestra un mensaje que les notifica el problema. La computadora no ejecutará un programa que contenga aunque sea sólo un error de sintaxis.

Por lo común, un programador desarrolla la lógica, escribe el código y compila el programa, recibiendo una lista de errores de sintaxis. Entonces los corrige y compila el programa de nuevo. La corrección del primer conjunto de errores con frecuencia revela otros nuevos que al principio no eran evidentes para el compilador. Por ejemplo, si usted usa un compilador en español y envía la declaración *El prro persiguen al gato*, el compilador al principio señalaría sólo un error de sintaxis. La segunda palabra, *prro*, es ilegal porque no forma parte del español. Sólo después de corregirla a *perro* el compilador hallaría otro error de sintaxis en la tercera palabra, *persiguen*, porque es una forma verbal incorrecta para el sujeto *perro*. Esto no significa que *persiguen* necesariamente sea la palabra equivocada. Quizá *perro* es incorrecto; tal vez el sujeto debería ser *perros*, en cuyo caso *persiguen* sería correcto. Los compiladores no siempre saben con exactitud qué quiere usted ni cuál debería ser la corrección apropiada, pero sí saben cuando algo anda mal con su sintaxis.

Cuando un programador escribe un programa tal vez necesite recompilar el código varias veces. Un programa ejecutable sólo se crea cuando el código no tiene errores de sintaxis. Después de que un programa se ha traducido al lenguaje de máquina, se guarda y puede ser ejecutado cualquier número de veces sin repetir el paso de traducción. Usted sólo necesita retraducir su código si hace cambios en las declaraciones de su código fuente. La figura 1-2 muestra un diagrama de este proceso en su totalidad.

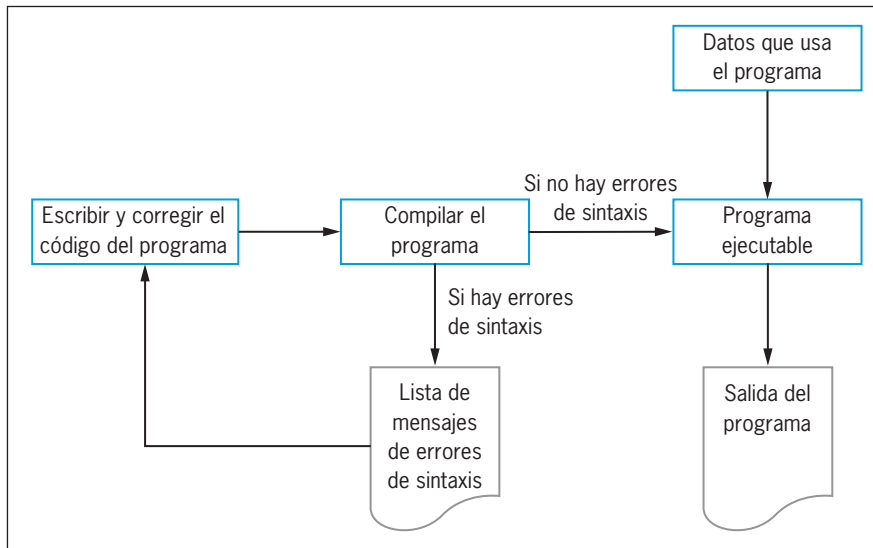


Figura 1-2 Creación de un programa ejecutable

Prueba del programa

Un programa que no tiene errores de sintaxis no necesariamente está libre de errores lógicos. Un error lógico resulta cuando se utiliza una declaración correcta desde el punto de vista sintáctico pero equivocada para el contexto actual. Por ejemplo, la declaración en español *El perro persigue al gato*, aunque sintácticamente es perfecta, no es correcta desde una perspectiva lógica si el perro persigue una pelota o si el gato es el agresor.

Una vez que un programa queda limpio de errores de sintaxis el programador puede probarlo, es decir, ejecutarlo con algunos datos de muestra para ver si los resultados son lógicamente correctos. Recuerde el programa para duplicar un número:

```
input myNumber
set myAnswer = myNumber * 2
output myAnswer
```

Si usted ejecuta el programa, proporciona el valor 2 como entrada para el mismo y se despliega la respuesta 4, ha ejecutado una corrida de prueba exitosa del programa.

Sin embargo, si se despliega la respuesta 40, quizá el programa contenga un error lógico. Tal vez usted tecleó mal la segunda línea del código con un cero extra, de modo que el programa se lee:

```
input myNumber
set myAnswer = myNumber * 20
output myAnswer
```

No lo haga

El programador tecleó 20 en lugar de 2.

Escribir 20 en lugar de 2 en la declaración de multiplicación causó un error lógico. Observe que desde el punto de vista sintáctico no hay nada incorrecto en este segundo programa (es

igual de razonable multiplicar un número por 20 que por 2) pero si el programador sólo pretende duplicar `myNumber`, entonces ha ocurrido un error lógico.

El proceso de hallar y corregir los errores del programa se llama **depuración**. Usted depura un programa al probarlo usando muchos conjuntos de datos. Por ejemplo, si escribe el programa para duplicar un número, luego introduce 2 y obtiene un valor de salida de 4, esto no necesariamente significa que el programa es correcto. Quizá tecleó por error este programa:

```
input myNumber
set myAnswer = myNumber + 2
output myAnswer
```

No lo haga
El programador tecleó
“+” en lugar de “*”.

Una entrada de 2 da como resultado una respuesta de 4, pero esto no significa que su programa duplique los números; en realidad sólo les suma 2. Si prueba su programa con datos adicionales y obtiene la respuesta errónea; por ejemplo, si introduce 7 y obtiene una respuesta de 9, sabe que hay un problema con su código.

La selección de los datos de prueba es casi un arte en sí misma y debe hacerse con cuidado. Si el departamento de recursos humanos desea una lista de los nombres de los empleados con antigüedad de cinco años, sería un error probar el programa con un pequeño archivo de muestra que sólo contiene empleados de tiempo indeterminado. Si los empleados más recientes no son parte de los datos que se usan para probar, en realidad no sabe si el programa los habría eliminado de la lista de cinco años. Muchas compañías no saben que su software tiene un problema hasta que ocurre una circunstancia extraña; por ejemplo, la primera vez que un empleado registra más de nueve dependientes, la primera vez que un cliente ordena más de 999 artículos al mismo tiempo o cuando a la internet se le agotan las direcciones IP asignadas, un problema que se conoce como *agotamiento IPV4*.

Poner el programa en producción

Una vez que se ha probado y depurado el programa en forma minuciosa, está listo para que la organización lo use. “Ponerlo en producción” significaría simplemente ejecutarlo una vez, si fue escrito para satisfacer una solicitud del usuario para una lista especial. Sin embargo, el proceso podría llevar meses si el programa se ejecutará en forma regular o si es uno de un gran sistema de programas que se están desarrollando. Quizá las personas que introducirán los datos deben recibir capacitación con el fin de preparar las entradas para el nuevo programa, los usuarios deben recibir instrucción para entender la salida o sea preciso cambiar los datos existentes en la compañía a un formato por completo nuevo para que tengan cabida en dicho programa. Completar la **conversión**, el conjunto entero de acciones que debe efectuar una organización para cambiar al uso de un programa o un conjunto de programas nuevos, en ocasiones puede llevar meses o años.

Mantenimiento del programa

Después de que los programas se colocan en producción, la realización de los cambios necesarios se denomina **mantenimiento**. Puede requerirse por diversas razones: por ejemplo, debido a que se han legislado nuevas tasas de impuestos, se alteró el formato de un archivo de entrada

o el usuario final requiere información adicional no incluida en las especificaciones de salida originales. Con frecuencia, la primera labor de programación que usted lleve a cabo requerirá dar mantenimiento a los programas escritos de manera previa. Cuando dé mantenimiento a los programas que otras personas han escrito apreciará el esfuerzo que hicieron para obtener un código claro, usar nombres de variables razonables y documentar su trabajo. Cuando hace cambios a los programas existentes repite el ciclo de desarrollo. Es decir, debe entender los cambios, luego planearlos, codificarlos, traducirlos y probarlos antes de ponerlos en producción. Si el programa original requiere una cantidad considerable de modificaciones podría ser retirado y empezaría el ciclo de desarrollo del programa para uno nuevo.

DOS VERDADES Y UNA MENTIRA

Comprensión del ciclo de desarrollo del programa

1. Entender el problema que debe resolverse puede ser uno de los aspectos más difíciles de la programación.
2. Las dos herramientas más comunes que se usan en la planeación de la lógica son los diagramas de flujo y el pseudocódigo.
3. La elaboración del diagrama de flujo de un programa es un proceso muy diferente si se usa un lenguaje de programación antiguo en lugar de uno más reciente.

La afirmación falsa es la número 3. A pesar de sus diferencias, los lenguajes de programación son bastante parecidos en sus capacidades básicas; cada uno puede manejar operaciones de entrada, procesamiento aritmético, operaciones de salida y otras funciones estándar. La lógica que se ha desarrollado para resolver un problema de programación puede ejecutarse usando cualquier cantidad de lenguajes.

Uso de declaraciones en pseudocódigo y símbolos de diagrama de flujo

Cuando los programadores planean la lógica para dar solución a un problema de programación con frecuencia usan dos herramientas: pseudocódigo o diagramas de flujo.

- El **pseudocódigo** es una representación parecida al inglés de los pasos lógicos que se requieren para resolver un problema. *Seudo* es un prefijo que significa *falso*, y *codificar* un programa significa ponerlo en un lenguaje de programación; por consiguiente, *pseudocódigo* simplemente significa *código falso*, o declaraciones que en apariencia se han escrito en un lenguaje de programación pero no necesariamente siguen todas las reglas de sintaxis de alguno en específico.
- Un **diagrama de flujo** es una representación gráfica de lo mismo.

Escritura enseudocódigo

Usted ha visto antes en este capítulo ejemplos de declaraciones que representan unseudocódigo y no hay nada misterioso en ellas. Las siguientes cinco declaraciones constituyen una representación enseudocódigo de un problema para duplicar un número:

```
start
  input myNumber
  set myAnswer = myNumber * 2
  output myAnswer
stop
```

Usar unseudocódigo implica escribir todos los pasos que se usarán en un programa. Por lo general, los programadores introducen suseudocódigo con una declaración inicial como `start` y lo terminan con uno de terminación como `stop`. Las declaraciones entre `start` y `stop` están en inglés y tienen una ligera sangría de modo que destaquen `start` y `stop`. La mayoría de los programadores no se preocupan por la puntuación como los puntos al final de las declaraciones delseudocódigo, aunque si usted prefiere ese estilo no sería un error usarlos. Del mismo modo, no hay necesidad de escribir con mayúscula la primera palabra en una declaración, aunque podría elegir hacerlo. Este libro sigue las convenciones de usar letras minúsculas para los verbos que comienzan las declaraciones en elseudocódigo y omitir los puntos al final de las mismas.

Elseudocódigo es bastante flexible porque es una herramienta de planeación y no el producto final. Por consiguiente, por ejemplo, quizá prefiera cualquiera de los siguientes:

- En lugar de `start` y `stop`, algunos desarrolladores deseudocódigo usan otros términos como `begin` y `end`.
- En lugar de `input myNumber` algunos escriben `get myNumber` o `read myNumber`.
- En vez de `set myAnswer = myNumber * 2`, algunos escribirán `calculate myAnswer = myNumber times 2` o `compute myAnswer as myNumber doubled`.
- En lugar de `output myAnswer`, muchos desarrolladores escribirán `display myAnswer`, `print myAnswer` o `write myAnswer`.

El punto es que las declaraciones enseudocódigo son instrucciones para recuperar un número original de un dispositivo de entrada y almacenarlo en la memoria, donde puede usarse en un cálculo, para después obtener la respuesta calculada de la memoria y enviarla a un dispositivo de salida de modo que una persona pueda verlo. Cuando al final usted convierte suseudocódigo en un lenguaje de programación específico, no tiene esta flexibilidad porque se requerirá una sintaxis determinada. Por ejemplo, si usa el lenguaje de programación `C#` y escribe la declaración para dar salida a la respuesta en un monitor, codificará lo siguiente:

```
Console.WriteLine(myAnswer);
```

El uso exacto de las palabras, las mayúsculas y la puntuación es importante en la declaración en `C#`, pero no en el deseudocódigo.

Trazo de diagramas de flujo

Algunos programadores profesionales prefieren escribir el pseudocódigo para trazar los diagramas de flujo debido a que este procedimiento es más parecido a escribir las declaraciones finales en el lenguaje de programación. Otros prefieren trazar diagramas de flujo para representar el flujo lógico debido a que éstos les permiten visualizar con más facilidad cómo se conectarán las declaraciones del programa. Los diagramas de flujo son una herramienta excelente, en especial para los programadores principiantes, pues son útiles a visualizar cómo se interrelacionan las declaraciones en un programa.

Usted puede trazar un diagrama de flujo a mano o usar software como Microsoft Word y Microsoft PowerPoint, que cuentan con herramientas para elaborarlos. Hay otros programas, como Visio y Visual Logic, específicamente para crear diagramas de flujo. Cuando usted crea uno dibuja formas geométricas que contienen las declaraciones individuales y que se conectan por medio de flechas. (En el apéndice B hay un resumen de todos los símbolos de los diagramas de flujo que verá en este libro.) Para representar un **símbolo de entrada** se usa un paralelogramo que indica una operación de entrada. Se escribe una declaración de entrada en inglés dentro del paralelogramo, como se muestra en la figura 1-3.

Las declaraciones de operaciones aritméticas son ejemplos de procesamiento. En un diagrama de flujo, se usa un rectángulo como el **símbolo de procesamiento** que contiene una declaración de procesamiento, como se muestra en la figura 1-4.

Para representar una declaración de salida se usa el mismo símbolo que para las de entrada: el **símbolo de salida** es un paralelogramo, como se muestra en la figura 1-5. Debido a que el paralelogramo se usa tanto para la entrada como para la salida, con frecuencia se llama **símbolo de entrada/salida** o **símbolo I/O**.



Algunos programas que usan diagramas de flujo (como Visual Logic) utilizan un paralelogramo inclinado hacia la izquierda para representar la salida. Siempre que el creador y el lector del diagrama de flujo estén en comunicación, la forma que se use es irrelevante. En este libro se seguirá la convención estándar de usar el paralelogramo inclinado hacia la derecha tanto para la entrada como para la salida.

A fin de mostrar la secuencia correcta de estas declaraciones se usan flechas o **líneas de flujo** para conectar los pasos. Siempre que sea posible, la mayor parte de un diagrama de flujo debe leerse de arriba hacia abajo o de izquierda a derecha en una página. Esta es la forma en que se lee el inglés, así que cuando los diagramas de flujo siguen esta convención son más fáciles de entender.

Para que un diagrama de flujo esté completo debe incluir dos elementos más: **símbolos terminales** o de inicio/fin en cada extremo. Con frecuencia usted coloca una palabra como **start** o **begin** en el primer símbolo terminal y una palabra como **end** o **stop** en el otro. Los símbolos terminales estándar tienen forma de pista de carreras; muchos programadores la llaman “pastilla” porque se parece a la forma del medicamento que se usa para aliviar una garganta irritada. La figura 1-6 muestra un diagrama de flujo completo para el programa que duplica un

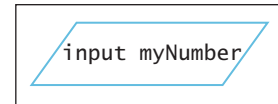


Figura 1-3 Símbolo de entrada

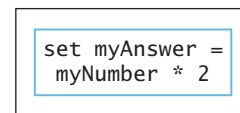


Figura 1-4 Símbolo de procesamiento

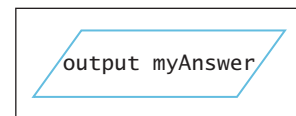


Figura 1-5 Símbolo de salida

número y elseudocódigo para el mismo problema. Es posible observar que las declaraciones en el diagrama de flujo y en elseudocódigo son las mismas, sólo el formato de presentación es diferente.

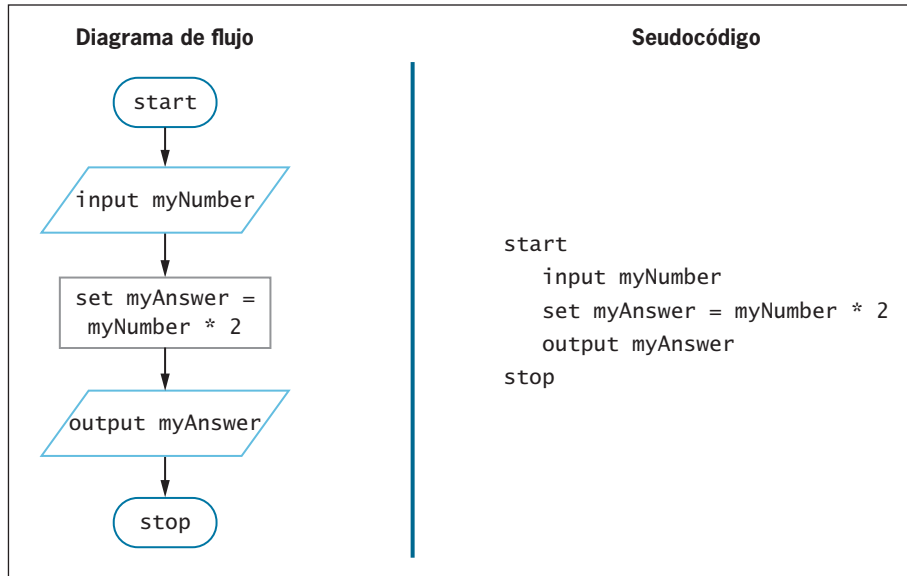


Figura 1-6 Diagrama de flujo yseudocódigo del programa que duplica un número

Los programadores rara vez crean unseudocódigo y un diagrama de flujo para el mismo problema. Por lo general se usa uno u otro. En un programa grande quizá usted prefiera escribir unseudocódigo para algunas partes y trazar un diagrama de flujo para otras.

Cuando usted indica a un amigo cómo llegar a su casa, podría escribir una serie de instrucciones o hacer un mapa. Elseudocódigo se parece a las instrucciones escritas paso a paso; un diagrama de flujo, como un mapa, es una representación visual de lo mismo.

Repetición de las instrucciones

Después de desarrollar el diagrama de flujo o elseudocódigo, el programador sólo necesita: 1) adquirir una computadora, 2) comprar un compilador de lenguaje, 3) aprender un lenguaje de programación, 4) codificar el programa, 5) intentar compilarlo, 6) arreglar los errores de sintaxis, 7) compilarlo de nuevo, 8) probarlo con varios conjuntos de datos y 9) ponerlo en producción.

Quizá en este momento usted piense: “¡Vaya! ¡Esto simplemente no vale la pena! ¿Todo ese trabajo para crear un diagrama de flujo o unseudocódigo y luego todos esos otros pasos? ¡Por 5 dólares puedo comprar una calculadora de bolsillo que duplicará cualquier número para mí en forma instantánea!”. Tiene razón. Si éste fuera un programa de computadora real y todo lo que hiciera fuera duplicar el valor de un número no valdría el esfuerzo. Escribir un programa tendría caso sólo si requiriera duplicar muchos números (digamos 10,000) en una cantidad de tiempo limitada (quizá los próximos dos minutos).

Por desgracia, el programa que se representa en la figura 1-6 no duplica 10,000 números; sólo duplica uno. Podría ejecutarlo 10,000 veces, por supuesto, pero esto requeriría que se sentara frente a la computadora y lo corriera una y otra vez. Se las arreglaría mejor con uno que pudiera procesar 10,000 números, uno después de otro.

18

Una solución es escribir el programa que se muestra en la figura 1-7 y ejecutar los mismos pasos 10,000 veces. Por supuesto, escribirlo requeriría mucho tiempo (también podría comprar la calculadora).

```
start
  input myNumber
  set myAnswer = myNumber * 2
  output myAnswer
  input myNumber
  set myAnswer = myNumber * 2
  output myAnswer
  input myNumber
  set myAnswer = myNumber * 2
  output myAnswer
  ...y así otras 9,997 veces más
```

No lo haga
Nunca desearía escribir una lista de instrucciones tan repetitiva.

Figura 1-7 Seudocódigo ineficiente para un programa que duplique 10,000 números

Una mejor solución es hacer que la computadora ejecute el mismo conjunto de tres instrucciones una y otra vez, como se muestra en la figura 1-8. La repetición de una serie de pasos se llama **ciclo**. Con este enfoque, la computadora obtiene un número, lo duplica, despliega la respuesta y luego comienza de nuevo con la primera instrucción. El mismo punto en la memoria, llamado `myNumber`, se reutiliza para el segundo número y para cualesquiera números subsiguientes. El punto en la memoria llamado `myAnswer` se reutiliza cada vez para almacenar el resultado de la operación de multiplicación. Sin embargo, la lógica que se ilustra en el diagrama de flujo de la figura 1-8 presenta un problema importante: la secuencia de instrucciones nunca termina. Esta situación se conoce en programación como **ciclo infinito**; un flujo repetitivo de lógica sin fin. Usted aprenderá cómo manejar este problema más adelante en este capítulo; estudiará un método más complejo en el capítulo 3.

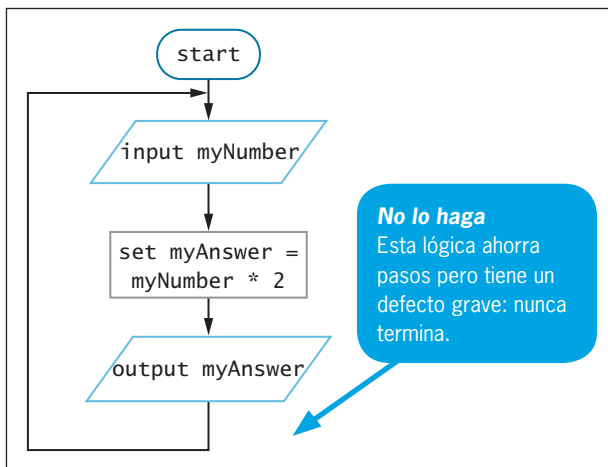


Figura 1-8 Diagrama de flujo de un programa de duplicación de números infinito

DOS VERDADES Y UNA MENTIRA

Uso de declaraciones de pseudocódigo y símbolos de diagrama de flujo

1. Cuando se traza un diagrama de flujo, se usa un paralelogramo para representar una operación de entrada.
2. Cuando se traza un diagrama de flujo, se usa un paralelogramo para representar una operación de procesamiento.
3. Cuando se traza un diagrama de flujo, se usa un paralelogramo para representar una operación de salida.

La afirmación falsa es la número 2. Cuando se traza un diagrama de flujo, se usa un rectángulo para representar una operación de procesamiento.

Uso de un valor centinela para terminar un programa

La lógica en el diagrama de flujo para duplicar números, que se muestra en la figura 1-8, tiene una falla importante: el programa contiene un ciclo infinito. Si, por ejemplo, los números de entrada se introducen por medio del teclado, el programa seguirá aceptando números y dando salida a sus valores duplicados para siempre. Por supuesto, el usuario podría negarse a teclear más números; pero el programa no puede avanzar más mientras esté esperando una entrada; mientras tanto, ocupa memoria de la computadora e inmoviliza recursos del sistema operativo. Dejar de introducir más números no es una solución práctica. Otra forma de terminar el

programa es simplemente apagar la computadora pero, una vez más, ésta no es la mejor solución ni una forma apropiada de hacerlo.

Una mejor forma es establecer un valor predeterminado para `myNumber` que signifique “¡Detén el programa!”. Por ejemplo, el programador y el usuario podrían acordar que este último nunca necesitará conocer el doble de 0 (cero), de modo que podría introducir un 0 para detenerlo. El programa entonces probaría cualquier valor contenido en `myNumber` que entre y si es 0 se detendría. Probar un valor también se llama **tomar una decisión**.

En un diagrama de flujo se representa una decisión al trazar un **símbolo de decisión**, que tiene forma de diamante. El diamante por lo general contiene una pregunta cuya respuesta es una de dos opciones mutuamente excluyentes, con frecuencia sí o no. Todas las buenas preguntas de computación tienen sólo dos respuestas mutuamente excluyentes, como sí y no o verdadero y falso. Por ejemplo, “¿Qué día es su cumpleaños?” no es una pregunta de computación adecuada porque hay 366 respuestas posibles. Sin embargo, “¿Su cumpleaños es el 24 de junio?” sí lo es porque la respuesta siempre es sí o no.

La pregunta para detener el programa de duplicación debería ser “¿El valor de `myNumber` que se acaba de introducir es igual a 0?” o “¿`myNumber` = 0?” para abreviar. El diagrama de flujo completo se verá entonces como el que se muestra en la figura 1-9.

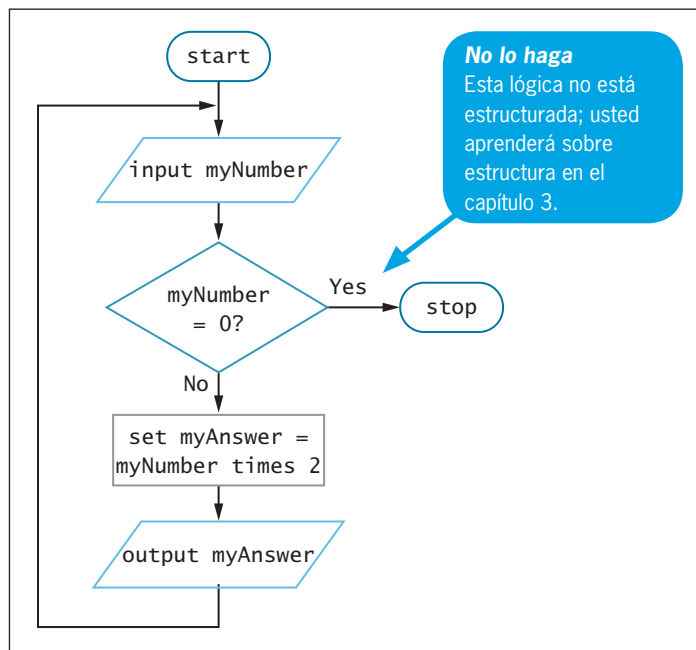


Figura 1-9 Diagrama de flujo del programa de duplicación de números con valor centinela de 0

Un inconveniente de usar 0 para detener un programa, por supuesto, es que no funcionará si el usuario necesita hallar el doble de 0. En este caso, podría seleccionar algún otro valor de

entrada de datos que nunca usará, como 999 o -1, para señalar que el programa debe terminar. Un valor predeterminado que detiene la ejecución de un programa con frecuencia se llama **valor comodín** porque no representa datos reales, sino sólo una señal para detener. En ocasiones, dicho valor se llama **valor centinela** debido a que representa un punto de entrada o salida, como un centinela que vigila una fortaleza.

No todos los programas dependen de la entrada de datos de un usuario desde un teclado; muchos leen los datos de un dispositivo de entrada, como un disco. Cuando las organizaciones guardan datos en un disco u otro dispositivo de almacenamiento en general no usan un valor comodín para señalar el final de un archivo. Por una parte, un registro de entrada podría tener cientos de campos y si almacena un registro comodín en cada archivo desperdiciará una gran cantidad de almacenamiento en “no datos”. Además, con frecuencia es difícil elegir valores centinela para los campos en los archivos de datos de una compañía. Cualquier `balanceDue`, incluso un cero o un número negativo, puede ser un valor legítimo, y cualquier `customer-Name`, incluso “ZZ”, podría ser el nombre de alguien. Por suerte, los lenguajes de programación reconocen el fin de los datos en un archivo de manera automática, por medio de un código que es almacenado en ese punto. Muchos lenguajes de programación usan el término **eof** (por el inglés *end of file* [*final del archivo*]) para referirse a este marcador que actúa en forma automática como un centinela. Este libro, por consiguiente, usa `eof` para indicar el final de los datos siempre que el uso de un valor comodín sea poco práctico o inconveniente. En el diagrama de flujo que se muestra en la figura 1-10, se sombrea la pregunta `eof`.

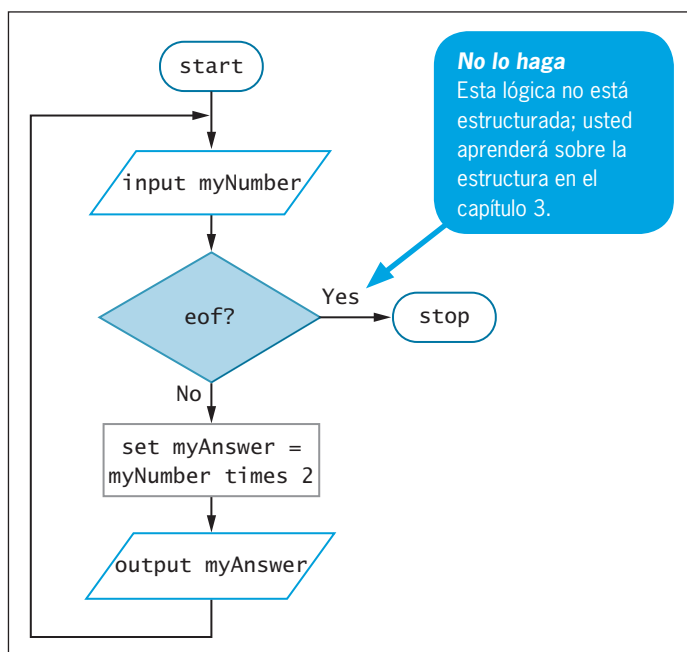


Figura 1-10 Diagrama de flujo que usa `eof`

DOS VERDADES Y UNA MENTIRA

Uso de un valor centinela para terminar un programa

1. Un programa que contiene un ciclo infinito nunca termina.
2. Un valor predeterminado que detiene la ejecución de un programa con frecuencia se llama valor comodín o valor centinela.
3. Muchos lenguajes de programación usan el término *fe* (por *file end* [*fin de archivo*]) para referirse a un marcador que actúa de manera automática como centinela.

La afirmación falsa es la número 3. El término *eof* (por *end of file*) es el término que más se usa para un centinela de archivo.

Comprensión de la programación y los ambientes del usuario

Es posible usar muchos enfoques para escribir y ejecutar un programa de computadora. Cuando usted planea la lógica de uno de ellos puede usar un diagrama de flujo, un pseudocódigo o una combinación de ambos. Cuando codifica el programa, puede teclear las declaraciones en una variedad de editores de texto. Cuando su programa se ejecuta podría aceptar entradas de un teclado, ratón, micrófono o cualquier otro dispositivo de entrada, y cuando proporcione la salida podría usar texto, imágenes o sonido. Esta sección describe los ambientes más comunes que encontrará como programador que recién inicia.

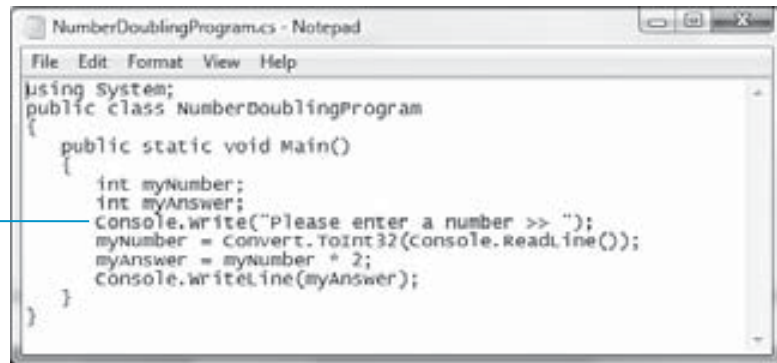
Comprensión de los ambientes de programación

Cuando usted planea la lógica para un programa de computadora puede usar papel y lápiz para crear un diagrama de flujo o software que le permita manipular las modalidades de dichos diagramas. Si decide escribir un pseudocódigo puede hacerlo a mano o con un programa de procesamiento de palabras. Para introducir el programa en una computadora de modo que lo traduzca y ejecute, por lo general usará un teclado para mecanografiar las declaraciones del programa en un editor. Puede hacerlo en uno de los siguientes:

- Un editor de texto sencillo
- Uno que sea parte de un ambiente de desarrollo integrado

Un **editor de texto** es un programa que se usa para crear archivos de texto sencillos. Es parecido a un procesador de palabras, pero sin tantas características. Usted puede usar alguno como Notepad, que está incluido en Microsoft Windows. La figura 1-11 muestra un programa C# en Notepad que acepta un número y lo duplica. Una ventaja de usar un editor de texto sencillo para mecanografiar y guardar un programa es que el programa completado no requiere mucho espacio del disco para almacenamiento. Por ejemplo, el archivo que se muestra en la figura 1-11 sólo ocupa 314 bytes.

Esta línea contiene un indicador que dice al usuario qué introducir. Usted aprenderá más sobre los indicadores en el capítulo 2.



```
using System;
public class NumberDoublingProgram
{
    public static void Main()
    {
        int myNumber;
        int myAnswer;
        Console.Write("Please enter a number >> ");
        myNumber = Convert.ToInt32(Console.ReadLine());
        myAnswer = myNumber * 2;
        Console.WriteLine(myAnswer);
    }
}
```

23

Figura 1-11 Un programa C# para duplicar números en Notepad

Usted puede usar el editor de un **ambiente de desarrollo integrado (IDE, *integrated development environment*)** para introducir su programa. Un IDE es un paquete de software que proporciona un editor, compilador y otras herramientas de programación. Por ejemplo, la figura 1-12 muestra un programa C# en el **Microsoft Visual Studio IDE**, un ambiente que contiene herramientas útiles para crear programas en Visual Basic, C++ y C#.

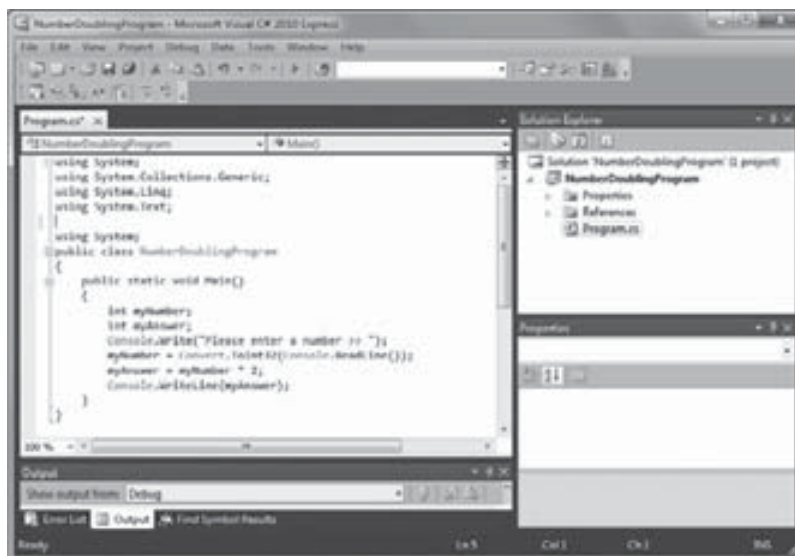


Figura 1-12 Un programa C# para duplicar números en Visual Studio

Usar un IDE es útil para los programadores porque por lo general proporciona características similares a las que se encuentran en muchos procesadores de palabras. En particular, un editor de IDE por lo común tiene características como las siguientes:

- Usa diferentes colores para desplegar varios componentes del lenguaje, lo que facilita la identificación de elementos como tipos de datos.
- Resalta errores de sintaxis en forma visual para usted.
- Emplea la terminación automática de las declaraciones; cuando empieza a teclear una el IDE sugiere una culminación probable, que usted puede aceptar con sólo oprimir una tecla.
- Proporciona herramientas que le permiten seguir paso a paso la ejecución de una declaración del programa a la vez de modo que puede seguir con más facilidad la lógica del mismo y determinar la fuente de cualquier error.

Cuando usa el IDE para crear y guardar un programa ocupa mucho más espacio en disco que cuando usa un editor de texto sencillo. Por ejemplo, el programa en la figura 1-12 ocupa más de 49,000 bytes de espacio de disco.

Aunque varios ambientes de programación podrían verse diferentes y ofrecer características distintas, el proceso para usarlos es muy parecido. Cuando usted planea la lógica para un programa usando un pseudocódigo o un diagrama de flujo, no importa cuál ambiente de programación utilice para escribir su código, y cuando escribe el código en un lenguaje de programación, no importa cuál ambiente use para hacerlo.

Comprensión de los ambientes de usuario

Un usuario podría ejecutar un programa que usted ha escrito en cualquier cantidad de ambientes. Por ejemplo, alguien podría hacerlo para duplicar números desde una línea de comandos como la que se muestra en la figura 1-13. Una **línea de comandos** es una ubicación en la pantalla de su computadora en la que usted teclea entradas de texto para comunicarse con el sistema operativo de la computadora. En el programa de la figura 1-13, se pide al usuario un número y se despliegan los resultados.

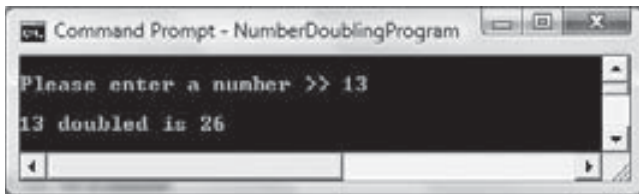


Figura 1-13 Ejecución de un programa para duplicar números en un ambiente de línea de comandos

Muchos programas no se ejecutan en la línea de comandos en un ambiente de texto, pero lo hacen usando una **interfaz gráfica del usuario**, o **GUI** (*graphical user interface*), que permite a los usuarios interactuar con un programa en un ambiente gráfico. Cuando se ejecuta un programa GUI, el usuario podría teclear entradas en un cuadro de texto o usar un ratón u otro dispositivo apuntador para seleccionar opciones en la pantalla. La figura 1-14 muestra un programa para duplicar números que ejecuta exactamente la misma tarea que el de la figura 1-13, pero con una GUI.



Figura 1-14 Ejecución de un programa para duplicar números en un ambiente GUI

Un programa de línea de comandos y uno GUI podrían escribirse en el mismo lenguaje de programación. (Por ejemplo, los programas que se muestran en las figuras 1-13 y 1-14 se escribieron con C#.) No obstante, sin importar cuál ambiente se use para escribir o ejecutar un programa, el proceso lógico es el mismo. Los dos programas en las figuras 1-13 y 1-14 aceptan entradas, ejecutan la multiplicación y la salida. En este libro, usted no se concentrará en saber cuál ambiente se usa para teclear las declaraciones de un programa ni se preocupará por el tipo de ambiente que verá el usuario. En cambio, se enfocará en la lógica que se aplica a todas las situaciones de programación.

DOS VERDADES Y UNA MENTIRA

Comprensión de la programación y los ambientes de usuario

1. Puede teclear un programa en un editor que es parte de un ambiente de desarrollo integrado, pero usar un editor de texto sencillo le proporciona más ayuda en la programación.
2. Cuando un programa corre o se ejecuta desde la línea de comandos, un usuario teclea el texto para proporcionar la entrada.
3. Aunque los ambientes GUI y de línea de comandos se ven diferentes, la lógica de entrada, procesamiento y salida se aplica a ambos tipos de programa.

La afirmación falsa es la número 1. Un ambiente de desarrollo integrado proporciona más ayuda de programación que un editor de texto sencillo.

Comprensión de la evolución de los modelos de programación

Las personas han escrito programas de computadora modernos desde la década de 1940. Los lenguajes de programación más antiguos requerían que los programadores trabajaran con direcciones de memoria y que memorizaran códigos incómodos asociados con los lenguajes de máquina. Los lenguajes de programación más recientes se parecen mucho más al lenguaje natural y son más fáciles de usar, en parte debido a que permiten a los programadores nombrar variables en lugar de usar direcciones de memoria poco manejables. Además, los

lenguajes de programación más novedosos permiten crear módulos o segmentos de programa autónomos que pueden armarse en diversas formas. Los programas de computadora más antiguos se escribían en una pieza, de principio a fin, pero los modernos rara vez se escriben así; son creados por equipos de programadores, y cada equipo desarrolla procedimientos de programa reutilizables y conectables. Escribir varios módulos pequeños es más fácil que escribir un programa grande, y la mayor parte de las tareas grandes son más fáciles cuando usted las divide para trabajar en unidades y hacer que otros colegas ayuden con algunas de ellas.



Ada Byron Lovelace predijo el desarrollo del software en 1843; con frecuencia se considera como la primera programadora. Las bases para la mayor parte del software moderno fueron propuestas por Alan Turing en 1935.

En la actualidad, los programadores usan dos modelos o paradigmas principales para desarrollar programas y sus procedimientos:

- La **programación procedimental** se enfoca en los procedimientos que crean los programadores. Es decir, los programadores procedimentales se centran en las acciones que se llevan a cabo; por ejemplo, obtener datos de entrada para un empleado y escribir los cálculos necesarios para generar un cheque de pago a partir de los datos. Los programadores procedimentales enfocarían la generación del cheque dividiendo el proceso en subtareas manejables.
- La **programación orientada hacia los objetos** se enfoca en los objetos o “cosas” y describe sus características (también llamadas atributos) y comportamientos. Por ejemplo, los programadores orientados hacia los objetos podrían diseñar una aplicación de nómina pensando en los empleados y cheques de pago, y describiendo sus atributos. Los empleados tienen nombres y números de seguro social y los cheques de pago contienen los nombres y las cantidades del cheque. Luego los programadores pensarán en los comportamientos de los empleados y los cheques de pago, como que los empleados obtienen aumentos y agregan dependientes y los cheques de pago son calculados y producidos. Los programadores orientados hacia los objetos construirán entonces aplicaciones a partir de estas entidades.

Con cualquier enfoque, procedimental u orientado hacia los objetos, usted puede generar un cheque de pago correcto y ambos modelos emplean módulos de programa reutilizables. La diferencia principal está en el enfoque que adopta el programador durante las primeras etapas de la planeación de un proyecto. Por ahora, este libro se enfoca en las técnicas de programación procedimental. Las habilidades que obtenga al aplicar este tipo de programación (declarar variables, aceptar entradas, tomar decisiones, producir salidas, etc.) le servirán en gran medida ya sea que al final escriba los programas con un enfoque procedimental, uno orientado hacia los objetos, o ambos. El lenguaje de programación en el que escriba su código fuente podría determinar su enfoque. Puede escribir un programa procedimental en cualquier lenguaje que soporte orientación a objetos, pero lo opuesto no siempre es cierto.

DOS VERDADES Y UNA MENTIRA

Comprensión de la evolución de los modelos de programación

1. Los programas de computadora más antiguos se escribían en muchos módulos separados.
2. Los programadores procedimentales se enfocan en las acciones que un programa lleva a cabo.
3. Los programadores orientados hacia los objetos se centran en los objetos de un programa y sus atributos y comportamientos.

La afirmación falsa es la número 1. Los programas más antiguos se escribían en una sola pieza; los más recientes se dividen en módulos.

Resumen del capítulo

- Juntos, el hardware (dispositivos físicos) y el software (instrucciones) realizan tres operaciones importantes: entrada, procesamiento y salida. Las instrucciones para la computadora se escriben en un lenguaje de programación que requiere una sintaxis específica; un compilador o intérprete traduce las instrucciones al lenguaje de máquina. Cuando la sintaxis y la lógica de un programa son correctas, usted puede correr o ejecutar el programa para obtener los resultados deseados.
- Para que un programa funcione en forma apropiada, usted debe desarrollar una lógica correcta. Es mucho más difícil localizar los errores lógicos que los de sintaxis.
- La labor de un programador implica entender el problema, planear la lógica, codificar el programa, traducirlo a lenguaje de máquina, probarlo, ponerlo en producción y mantenerlo.
- Cuando los programadores planean la lógica de una solución para un problema de programación con frecuencia usan diagramas de flujo o pseudocódigo. Cuando usted traza un diagrama de flujo usa paralelogramos para representar las operaciones de entrada y salida, y rectángulos para representar el procesamiento. Los programadores también toman decisiones para controlar la repetición de los conjuntos de instrucciones.
- Para evitar la creación de un ciclo infinito cuando usted repite las instrucciones puede probar un valor centinela. Se representa una decisión en un diagrama de flujo al dibujar un símbolo en forma de diamante que contiene una pregunta cuya respuesta es sí o no.
- Usted puede teclear un programa en un editor de texto sencillo o uno que sea parte de un ambiente de desarrollo integrado. Cuando los valores de datos de un programa se introducen desde un teclado, pueden ingresarse en la línea de comandos en un ambiente de texto o en una GUI. De cualquier forma, la lógica es similar.

- Los programadores procedimentales y orientados hacia los objetos enfocan los problemas de manera diferente. Los procedimentales se concentran en las acciones que se ejecutan con los datos. Los orientados hacia los objetos se enfocan en los objetos, sus comportamientos y atributos.

Términos clave

Un **sistema de cómputo** es una combinación de todos los componentes que se requieren para procesar y almacenar datos usando una computadora.

El **hardware** es el conjunto de dispositivos físicos que componen un sistema de cómputo.

El **software** consiste en los programas que indican a la computadora qué debe hacer.

Los **programas** son los conjuntos de instrucciones para una computadora.

La **programación** es el acto de desarrollar y escribir programas.

El **software de aplicación** comprende todos los programas que usted aplica a una tarea.

El **software de sistema** comprende los programas que usted usa para manejar su computadora.

La **entrada** describe la introducción de elementos de datos en la memoria de la computadora por medio de los dispositivos de hardware como teclados y ratones.

Los **elementos de datos** incluyen todo el texto, los números y otra información procesada por una computadora.

El **procesamiento** de elementos de datos implica organizarlos, comprobar su precisión o realizar operaciones matemáticas en ellos.

La **unidad central de procesamiento**, o **CPU**, es el componente de hardware que procesa los datos.

La **salida** describe la acción de recuperar información de la memoria y enviarla a un dispositivo, como un monitor o impresora, de modo que las personas puedan ver, interpretar y trabajar con los resultados.

Información son los datos procesados.

Los **dispositivos de almacenamiento** son los tipos de equipo de hardware, como discos, que contienen información para su recuperación posterior.

Los **lenguajes de programación**, como Visual Basic, C#, C++, Java o COBOL, se usan para escribir los programas.

El **código de programa** es el conjunto de instrucciones que un programador escribe en un lenguaje de programación.

Codificar el programa es la acción de escribir instrucciones en lenguaje de programación.

La **sintaxis** de un lenguaje son sus reglas gramaticales.

Un **error de sintaxis** es un error en el lenguaje o la gramática.

La **memoria de la computadora** es el almacenamiento interno temporal dentro de una computadora.

La **memoria de acceso aleatorio (RAM)** es el almacenamiento interno temporal de la computadora.

El término **volátil** describe el almacenamiento cuyo contenido se pierde cuando la energía eléctrica se interrumpe.

El término **no volátil** describe el almacenamiento cuyo contenido se conserva cuando la energía eléctrica se interrumpe.

El **lenguaje de máquina** es un lenguaje de circuitería encendido/apagado de una computadora.

El **código fuente** son las declaraciones que un programador escribe en un lenguaje de programación.

Código objeto es lenguaje de máquina traducido.

Un **compilador o intérprete** traduce un lenguaje de alto nivel a uno de máquina e indica si ha usado en forma incorrecta un lenguaje de programación.

El **lenguaje binario** se representa usando una serie de 0 (ceros) y 1 (unos).

Correr o ejecutar un programa significa que se llevan a cabo sus instrucciones.

Los **lenguajes de programación interpretados** (también llamados **lenguajes de programación de scripting** o **lenguajes de script**) como Python, Lua, Perl y PHP se usan para escribir programas que se introducen en forma directa desde un teclado. Los lenguajes de programación interpretados se almacenan como texto y no como archivos ejecutables binarios.

Un **error lógico** ocurre cuando se ejecutan instrucciones incorrectas, o cuando éstas se efectúan en el orden incorrecto.

Usted desarrolla la **lógica** del programa de computadora cuando da instrucciones a la computadora en una secuencia específica, sin omitir alguna instrucción ni agregar instrucciones superfluas.

Una **variable** es una ubicación de memoria nombrada cuyo valor puede variar.

El **ciclo de desarrollo del programa** consiste en los pasos que se siguen durante la vida de un programa.

Los **usuarios** (o **usuarios finales**) son personas que emplean los programas de computadora y obtienen beneficios de ellos.

La **documentación** consiste en todo el papeleo de soporte para un programa.

Un **algoritmo** es la secuencia de pasos necesarios para resolver cualquier problema.

Una **gráfica IPO** es una herramienta de desarrollo de programas que define las tareas de entrada, procesamiento y salida.

Una **gráfica TOE** es una herramienta de desarrollo de programas que lista tareas, objetos y eventos.

Prueba de escritorio es el proceso de recorrer la solución de un programa en papel.

Un **lenguaje de programación de alto nivel** soporta sintaxis en inglés.

Un **lenguaje de máquina de bajo nivel** está formado por 1 (unos) y 0 (ceros) y no usa nombres de variables que se interpretan con facilidad.

Depuración es el proceso de hallar y corregir errores del programa.

Conversión es el conjunto entero de acciones que debe emprender una organización para cambiar al uso de un programa o conjunto de programas nuevos.

El **mantenimiento** consiste en todas las mejoras y correcciones hechas a un programa después de que está en producción.

El **seudocódigo** es una representación en inglés de los pasos lógicos que se requieren para resolver un problema.

Un **símbolo de entrada** indica una operación de entrada y en los diagramas de flujo se representa con un paralelogramo.

Un **símbolo de procesamiento** indica una operación de procesamiento y en los diagramas de flujo se representa con un rectángulo.

Un **símbolo de salida** indica una operación de salida y en los diagramas de flujo se representa con un paralelogramo.

Un **símbolo de entrada/salida** o **símbolo I/O** en los diagramas de flujo se representa con un paralelogramo.

Las **líneas de flujo**, o flechas, conectan los pasos en un diagrama de flujo.

Un **símbolo terminal** indica el inicio o el fin de un segmento de diagrama de flujo y se representa con una pastilla.

Un **ciclo** es una repetición de una serie de pasos.

Un **ciclo infinito** ocurre cuando la lógica que se repite no puede terminar.

Tomar una decisión es el acto de probar un valor.

Un **símbolo de decisión** tiene forma de diamante y en los diagramas de flujo se usa para representar una decisión.

Un **valor comodín** es un valor predeterminado que detiene la ejecución de un programa.

Un **valor centinela** es un valor predeterminado que detiene la ejecución de un programa.

El término **eof** significa *fin del archivo*.

Un **editor de texto** es un programa que se usa para crear archivos de texto sencillos; es parecido a un procesador de palabras, pero sin tantas características.

Un **ambiente de desarrollo integrado (IDE)** es un paquete de software que proporciona un editor, un compilador y otras herramientas de programación.

Microsoft Visual Studio IDE es un paquete de software que contiene herramientas útiles para crear programas en Visual Basic, C++ y C#.

Una **línea de comandos** es una ubicación en la pantalla de su computadora en la que teclea entradas de texto para comunicarse con el sistema operativo de la computadora.

Una **interfaz gráfica del usuario**, o **GUI**, permite a los usuarios interactuar con un programa en un ambiente gráfico.

La **programación procedimental** es un modelo de programación que se enfoca en los procedimientos que crean los programadores.

La **programación orientada hacia los objetos** es un modelo de programación que se enfoca en objetos, o “cosas”, y describe sus características (también llamadas atributos) y comportamientos.